Carbon-Aware Memory Placement

Anonymous Author(s)

ABSTRACT

The carbon footprint of software activities is determined by embodied and operational emissions of hardware resources. This paper presents `carbon-FooBar`[1], a concept that enables operating systems to make carbon-aware memory placement decisions.

Main memory has become heterogeneous in today's computer systems. In addition to traditional (and volatile) main memory (e.g. DRAM), novel memory technologies with persistent properties are often also available (e.g. PRAM, FRAM, MRAM). Complementary, there are a large number of new memory interfaces (e.g. high-bandwidth, graphics, and low-power memory) that have to be additionally taken into account by the operating system when allocating memory. The availability of new memory technologies and interfaces enables systems with improved energy efficiency. At the same time, the new memory interfaces have revealed serious flaws in the current state-of-the-art memory abstractions in operating systems. Hence, moving away from the homogeneous perspective of memory resources is a crucial step towards significantly reducing the energy consumption and ultimately the carbon footprint of today's computer systems.

With `carbon-FooBar`, we propose an approach that combines information on characteristics of (i) active workloads and (ii) available memory resources with a carbon model. `carbon-FooBar` transforms the combined information into memory placement decisions at operating system level. The placement decisions that are made result in improved operating conditions (i.e. better energy efficiency and lower carbon footprint) for the available storage resources at the hardware level.

1 INTRODUCTION

Recent years have brought a rapid expansion of the range and variety of available main-memory technologies and architectures. When previously, evolutionary generations of DDR DRAM were the staple in the main-memory tier, today, system integrators can draw from a much-widened design space, including significant interface variations (GDDR, LPDDR, HBM) to fundamentally the same DRAM technology, up to completely new memory cell types (PRAM, FRAM, MRAM). Under such conditions, the long-held tenet of abstracting physical memory resources into a homogeneous virtual memory space visible to the programmer is beginning to show severe shortcomings [6, 19, 20].

Until now, memory subsystems received much less attention from efforts to improve energy efficiency—and thus linked operational carbon-impact—than active compute resources, even though memory chips and interconnects can easily draw power in the same order of magnitude as a CPU [17]. Furthermore, the *embodied carbon emissions*, both for acquisition of novel memory technologies, as well as their limited life-time like prominently in the case of NVRAM, is not sufficiently captured by existing programming and operating system abstractions. In light of the societal importance

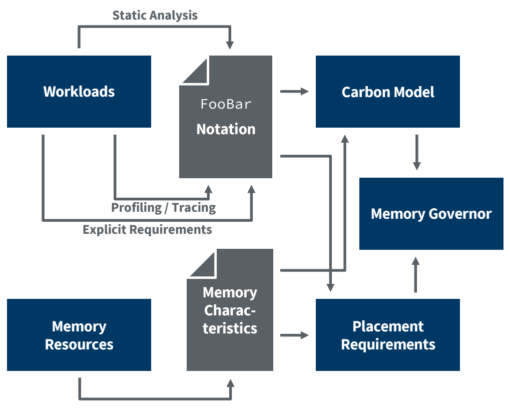

Figure 1: Architecture of our carbon-aware memory placement approach `carbon-FooBar`.

of economic and ecologic concerns in computing technology, it is crucial to leverage each memory technology to its fullest potential to meet both performance and carbon-efficiency objectives.

Thus, this paper presents `carbon-FooBar`, a concept that enables operating systems to make carbon-aware memory placement decisions. The operating system and system software offer the necessary interfaces to improve and extend existing abstractions and mediate between programmer and hardware perspectives. `carbon-FooBar` refrains from burdening programmers with an unfiltered view of raw memory-resource characteristics. Instead, the flow of information is inverted by giving systems software explicit or implicit knowledge of workload in addition to memory resource characteristics. This establishes a common point where carbon-efficient memory placement decisions can be made.

This paper makes the following four main contributions:
(i) the `carbon-FooBar` approach to collect and apply necessary information for carbon-aware memory placement decisions. This includes analysis methods at *development time*, profiling at *runtime*, as well as, characteristics of memory resources at *system-setup time*.
(ii) the design of a carbon model to determine the embodied and operational carbon emissions of memory allocations.
(iii) the design of a operating-system component, Memory Governor, for carbon-aware memory placements.
(iv) a case study for different types of memory allocations (i.e. different lifetime and access patterns) and types of memory resources.

The rest of this paper is structured as follows. Section 2 discusses `energy-FooBar`, the underlying concept for `carbon-FooBar`. Section 3 outlines the design of the Memory Governor and Section 4 discusses placement requirements for carbon-aware memory placement. We discuss the carbon model of `carbon-FooBar` and a case study of different memory types in Section 5 and Section 6, respectively. Section 7 gives an outlook on future work, Section 8 discusses related work, and Section 9 concludes this paper.

[1]name changed for blind review

2 THE ENERGY-FOOBAR APPROACH

The trend to compose machines from heterogeneous memory technologies poses new challenges to systems software. On the one hand, overly generic placement policies fail to capture the optimisation potential. On the other hand, system operators can not be expected to customise placement strategies for each machine under their responsibility. However, existing placement strategies only have limited means (e.g. NUMA domains) to cope with this new heterogeneity. In particular, they miss empirical analysis mechanisms to adapt to evolving workload and resource conditions.

The scope of `energy-` and `carbon-FooBar` are machines with one common physical address space and a heterogeneous set of memory technologies. However, scenarios beyond this scope, for example with disaggregated memory or RDMA, are discussed in Section 7. One key design decision is the granularity at which placement decisions are optimised. Optimising at a very fine-grained level (e.g. single memory accesses) may theoretically yield the best results but comes with high overheads. Vice versa, optimising only at coarse granularity (e.g. virtual address spaces) has little overhead but also limits the optimisation potential. We believe that optimising at *buffer granularity* best fits existing programming paradigms. As developers are experts in the dynamics of their applications, they tend to organise their data into objects or buffers (either by manual allocation or as part of the programming environment).

Our envisioned architecture is illustrated in Figure 1. The left side shows the empirical analyses, which generate characterisations of workloads (top) and the memory resources (bottom). These characterisations are used by the tools on the right side (i.e. carbon model and placement requirements), which aid the Memory Governor to make placement decisions. The characterisation of memory resources includes their performance and energy behaviour during representative access patterns. Once collected, this information is utilised by the carbon model to predict the energy demand and carbon emissions. Conceptually, the workload characterisation consists of three parts: a) explicit requirements at development time, b) static analysis ahead of runtime, and c) profiling/tracing at runtime.

Explicit Requirements. We believe it is beneficial to allow developers to share their specialised knowledge of workload behaviour. Therefore, we provide the possibility to explicitly state functional (e.g. persistent vs. volatile) and non-functional (e.g. latency bounds) requirements. In order to retain development efficiency and backwards compatibility, this source of information is optional and thought for especially critical application data structures.

Static Analysis. Although only capable of capturing memory behaviour known ahead of runtime, static code analysis can yield essential information about memory allocations. The analysis can be conducted once and later reused for every workload start.

Profiling/Tracing. Workload behaviour at runtime is observed by means of performance monitoring counters (PMCs). This allows our system to respond to changing workload conditions.

The three different information sources are consolidated into a unified workload characterisation using the `energy-FooBar` notation, which is designed to describe workload behaviour. This characterisation is used to to determine possible placement locations and the carbon footprint of a placement. Eventually, the Memory Governor uses these information for memory placement decisions.

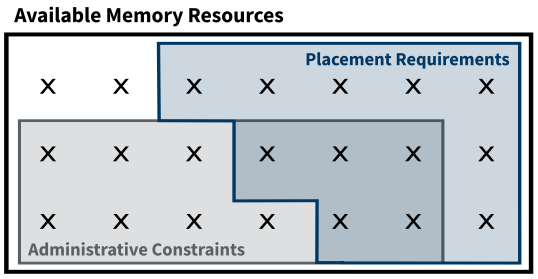

Figure 2: Memory Governor decision space.

3 CARBON-AWARE MEMORY GOVERNOR

The *Memory Governor* is the core component of the `carbon-FooBar` approach. It is part of the operating system and responsible to automatically make efficient memory placement decisions. The name Memory Governor follows the example of similar components managing global strategies for shared resources. For example Linux implements a CPU frequency governor, which globally balances the CPU-time demands of all running applications and optimises the CPU frequency according to an optimisation goal (e.g. energy efficiency). In the Memory Governor case that means, managing the globally available memory resources and responding to memory allocations in a carbon-efficient way.

Figure 2 visualises the underlying concept. The black rectangle comprises all available memory resources. Thereby, each cross represents a single resource (or part of a resource). Due to placement constraints (blue), only parts of the globally available memory resources can be considered for a memory request. Placement constraints can be, for example, depending on functional properties like non-volatile byte-addressable memory (NVRAM). The decision space can be further restricted by administrative constraints (grey) imposed by system operators.

In order to fulfil its task, the Memory Governor needs to keep track of the available and free memory resources. For each memory request it calculates the possible placements, that is, the intersection of placement and administrative constraints. The remaining possible placements constitute the decision space for the Memory Governor, in which it tries to satisfy its target (i.e. carbon efficiency) in a best-effort approach. It can therefore utilise the *sensitivity* of a workload to a specific hardware resource (see Section 4).

Due to this design and by implementing the Memory Governor within the operating system, it can also limit the wear and tear of memory resources with limited lifetimes (such as NVRAM). As the carbon emissions to manufacture memory (embodied carbon) can constitute a significant amount of the total carbon emissions, this is necessary to be truly carbon aware. As the Memory Governor has a system-wide view on all resources and the carbon intensity of the current power supply, it can balance all of these constraints. A second reason for implementing the Memory Governor within the operating system is the observation that memory allocations occur frequently. Hence, the Memory Governor requires an efficient implementation. Existing implementations of model-based placement strategies [21] show that efficient and practical implementations are possible, and provide a starting point.

4 PLACEMENT REQUIREMENTS

Communicating placement requirements from development time to runtime requires a standardised exchange format. To interface between workloads and the Memory Governor, we need to transform *workload behaviour* and *sensitivity* to resource characteristics into placement requirements. In this context, sensitivity stands for how a workload will react to a memory resource with specific functional and non-functional properties. Our proposed interface is twofold: an *Application Programming Interface* (API) for explicit specification on an allocation granularity for developers, as well as, a mechanism to deduce implicit sensitivity from observed workload behaviour.

Instead of mere library calls to an explicit API, we propose a more accessible integration with existing programming language features such as function decorators or scoped allocators to offer a low-overhead specification on how to serve specific buffers. This way, requirements and changing sensitivity can be denoted for a limited section of the application source code. They can also be inherited over source-code sections and can be lifted automatically as the defined scope is left.

Listing 1 provides a brief example of the envisioned API usage with scoped allocators in C++, where different memory requirements, such as `latency` and `randomAccess`, are expressed, reflecting their performance impact on the respective buffer.

```
AllocCharacteristic LatencyNVSpec {
  // uninitialised fields default to 0.f or false,
  weight_latency = 3.f,
  weight_randomAccess = 1.f,
  non_volatile = true
};

{
  // all allocations in this scope are
  // associated with LatencyNVSpec
  MemGuard guard(LatencyNVSpec);
  Index * jumpListA = new Index[4096];
}
```

Listing 1: Example of the proposed API for specifying desired memory characteristics explicitly using scoped allocators.

By overwriting the standard memory-allocation functions, the proposed API tracks each distinct memory allocation. The allocation is then identified as a logical buffer and associated with the currently active memory requirement set. The identified buffers and the per-buffer resource requirements are passed to the Memory Governor using sensitivity weights to specify the priority for independent characteristics. In the early stages of the Memory Governor implementation, the explicit API affords an early validation point, as the applicability of the chosen requirements and sensitivity format can be evaluated without relying on sophisticated automatic mechanisms for an implicit deduction. The second step, however, comprises the automatic transformation of the behavioural data about workloads gathered during runtime. Said behaviour model is build based on benchmarks, which are tailored to fill in the parameters like latencies and access granularities as well as the read and write dynamics of memory technology and controller. Using profiling, a previously unknown workload can be classified along those parameters to match sensitivity classes. To that end, *static code analysis* can be another ways to recognise pre-defined classes, although the expected complexity is much higher.

5 MEMORY CARBON MODEL

The carbon footprint of memory accesses is the basis for carbon-aware and carbon-efficient memory placement. The memory carbon model estimates the carbon emissions of memory placements depending on the memory-access behaviour and utilised hardware. We identify two use cases for these estimations relevant to this work: a) as a tool for developers to analyse their software's memory-related carbon footprint ahead of runtime. b) as an integrated component within the operating system at runtime whose estimations are passed to the Memory Governor.

With the `energy-FooBar` notation and memory characteristics as input, the model derives the carbon footprint of placement decisions. Based on the model's operational and embodied carbon estimations for different options, the Memory Governor determines the most carbon-efficient placement. Central to the carbon model are two parts: a) the carbon metric used for calculating the carbon footprint associated of a memory placement and b) the energy model used for calculating operational emissions of memory accesses. Section 5.1 introduces the carbon metric utilised by the carbon model: the Software Carbon Intensity (SCI) specification [12]. The underlying energy model is presented in Section 5.2.

5.1 Carbon Metric

As a metric for assessing the carbon footprint of memory placement decisions, the carbon model utilises the Software Carbon Intensity (SCI) specification [12]. The SCI was suggested by the Greensoftware Foundation as a standard metric for the carbon footprint of software and is currently under review at ISO. The SCI takes into account both operational carbon emissions (O) and embodied emissions of the hardware (M). The carbon footprint is then derived per application-specific units of work (R):

$$SCI = \frac{O + M}{R} \tag{1}$$

Operational Carbon. The operational carbon emissions of memory placements are determined by the operational energy demand E of memory accesses and the carbon intensity I of the energy supply. The SCI, thus, expresses the operational carbon emissions as

$$O = E \cdot I \tag{2}$$

Deriving the operational energy of memory technologies under different access patterns requires the use of a separate energy model, which is further discussed in Section 5.2. The carbon intensity of the available energy mix has to be constantly monitored and provided as input for the carbon model. These values can either be obtained from the local energy providers or from providers such as ElectricityMap [10] that analyse the carbon intensity per country.

Embodied Carbon. For a holistic view of the carbon footprint of memory placement strategies, the carbon model also considers emissions related to production and disposal of hardware. The SCI proposes the following equation for embodied carbon emissions:

$$M = TE \cdot TS \cdot RS \tag{3}$$

The embodied carbon emissions attributed to memory placements are, thus, determined by the following factors: The memory's total embodied emissions (TE). The share of the memory's lifespan taken up by the memory placement (TS). The share of the available resources (RS). The total embodied emissions of a given memory component can be obtained from hardware vendors.

Considering both *TS* and *RS*, the carbon model differentiates between two classes of memories: a) wear-sensitive memories and b) wear-agnostic memories. Wear-sensitive memories show limited endurance. Due to their physical characteristics, these memory technologies are expected to fail after a number of accesses. Thus, each access can be attributed a proportional share of the total embodied emissions. For example, flash-bashed and phase-change–based memories such as Intel Optane can be classified as such memories [2, 5]. *TS* is, therefore, determined by the ratio of the memory accesses related to a placement decision and the overall available accesses until expected failure. The number of memory accesses related to the placement of a buffer is obtained from static or dynamic analyses (see Section 4). As the wear affects the entire memory resource (e.g. a DIMM) *RS* equals one.

The second class, wear-agnostic memories, is not limited in their expected lifespan by individual memory accesses. Memories like DRAM are typically expected to outlive their system. Therefore, their expected lifespan is set to the system's anticipated lifespan. With this class of memories, *RS* equals the share of the memory capacity in use for the memory placement.

5.2 Energy Model

An integral part of determining the operational carbon emissions is evaluating the energy behaviour of the underlying hardware. This behaviour is incorporated into our memory carbon model in form of an energy model. The model's input consists of the following:

Memory Access Behaviour. The energy demand of using memory heavily depends on how the memory is accessed. Therefore, the energy model uses the `energy-FooBar` notation to represent a workload's memory access behaviour (e.g. determined by the runtime-behaviour monitoring as described in Section 2).

Allocation Granularity. The Memory Governor is required to make carbon-efficient memory placement decisions at different granularities. For example, what are the costs to place a huge vs. small buffer in NVM memory? How do the costs change when HBM memory is used? To account for the difference in scope and granularity, the energy model retrieves the granularity as input.

Hardware. The energy demand, and therefore the energy model, is highly hardware specific and receives hardware characteristics as input. We use a top-down approach to combine general models and precise hardware-specific models. Therefore, we create general models for similar hardware, for example, one for NVRAM and one for HBM. These general models can be refined into a specific models (e.g. the NVRAM model into a PC-RAM model).

The implementation of resource models in general and energy models in particular often utilises machine-learning techniques. Both, simple linear [15] and ensemble models [26], as well as sophisticated techniques based on neural networks [16], have shown great results for resource and energy models. These techniques differ in expressiveness and accuracy on the one side and training and execution costs on the other side.

6 CASE STUDY

To illustrate the decision space of the proposed Memory Governor, we analyse an exemplary system that contains 3 different memory technologies: DRAM, Viking NVDIMMs, and Intel Optane. The device is equipped with 256 GB memory of each type. We determine how the carbon intensity of various workloads changes in environments with different energy supplies (*I*). Table 1 and Table 2 lists the operational and embodied emissions, respectively.

[2]data based on flash memory

	DRAM	Viking NVDIMM	Optane[2]
embodied CO_2e/GB	313 g	386 g	73 g

Table 1: Embedded carbon cost for different memory classes.

Energy source	DRAM	Viking NVDIMM	Optane[2]
Wind	44 pg	44 pg	16 pg
Natural Gas	1880 pg	1880 pg	685 pg
Coal	3920 pg	3920 pg	1428 pg
Mix	1472 pg	1472 pg	536 pg

Table 2: Operational carbon cost to access 4 kB of memory.

6.1 Carbon Emissions

Embodied Carbon. The embodied carbon in DRAM is well understood. Both, the authors in [30] and [13] report around 5.0 kg CO_2e for 16 GB DRAM. In contrast, only little information on the embodied carbon of NVM is available. In the absence of any lifecycle assessments of Intel Optane and Viking NVDIMMs, we estimate their properties using related components. For Intel Optane, we use an SSD as substitute. The embodied carbon for a 256 GB SSD varies between 50 kg CO_2e [29], 18.7 kg CO_2e [30], and 7.7 kg CO_2 [13][3]. For this case study, we use the median of those sources, which is 18.7 kg CO_2e. Viking NVDIMMs consist of DRAM that is written to an SSD in case of an outage. As such, we estimate the embodied carbon as the sum of the costs for the DRAM and SSD of equal size.

Operational Carbon. For DRAM, we assume an energy consumption of 0.4 W/GB. Since RAM needs to be constantly refreshed, we attribute that amount of power consumption for the entire lifetime of the allocation. SSDs have a negligible power consumption while they are not being accessed and thus we attribute operational carbon to a workload on a per-access basis. The carbon emitted by a single access is denoted in Table 2. Viking NVDIMMs only access the DRAM during normal operation and only write to the SSD in the event of a power failure. As a result, we treat this kind of memory like DRAM when calculating operational carbon.

For our calculations we use four different energy sources. Wind energy (11 g CO_2e/kWh [8]), natural gas (470 g CO_2e/kWh [22]), coal (980 g CO_2e/kWh [32]), and a mix consisting of half renewable energy (wind) and half fossil fuels (25 % coal and 25 % gas).

6.2 Workload Analysis

Using the base emissions determined above, we study the emissions of three different workloads. We show that the ideal placement for allocations both depends on the type of workload and the current carbon intensity of the energy supply. Each workload allocates a resource share (*RS*) of 4 kB ($RS = 1.6 \cdot 10^{-8}$ for all cases). The workloads differ, however, in how long the memory is allocated and their access pattern. DRAM is treated as wear-agnostic, while Intel Optane is wear-sensitive. Viking NVDIMMs only write to the SSD on a power failure, so they are treated as wear-agnostic. For

[3]no CO_2 equivalent emissions are provided

wear-agnostic memory, we assume a lifetime of five years. For Intel Optane, we assume an endurance of 150 TB written.

Scenario A: Short-Lived Allocation, Single Access: This workload allocates the memory for 10 μs and accesses it only once. There are no placement requirements for this allocation. This results in a proportionate life span of $TS = 6.3 \cdot 10^{-14}$ for wear-agnostic memory and $TS = 2.7 \cdot 10^{-11}$ for wear-sensitive devices. For this type of allocation, putting the allocation into DRAM is the optimal placement strategy for all energy types. In the energy mix the allocation only emits $1.6 \cdot 10^{-15}$ g CO_2e when placed in DRAM, while it would emit $5.4 \cdot 10^{-10}$ g CO_2e when placed in Intel Optane.

Scenario B: Medium-Lived Allocation, Many Accesses: This workload allocates the memory for 50 ms and accesses it 100 000 times. The placement is restricted to persistent memory. This results in $TS = 3.2 \cdot 10^{-10}$ for wear-agnostic memory and $TS = 2.7 \cdot 10^{-6}$ for wear-sensitive devices. Since this allocation requires placement in a non-volatile memory, placing it in DRAM is not an option. Due to the wear-sensitive nature of Optane it is highly inefficient for this kind of memory placement (5.3 g CO_2e when using the energy mix), making the Viking NVDIMM ($8.2 \cdot 10^{-12}$ g CO_2e using the energy mix) the best option across all energy sources.

Scenario C: Long-Lived Allocation, Few Accesses: This workload allocates the memory for 5 s and accesses it five times. There are no placement requirements for this allocation. This results in $TS = 3.2 \cdot 10^{-8}$ for wear-agnostic memory and $TS = 1.3 \cdot 10^{-10}$ for wear-sensitive devices. For this type of allocation, the ideal placement depends on the current energy source. DRAM is the ideal placement decision for this allocation when using wind energy with $6.4 \cdot 10^{-11}$ g CO_2e ($8.0 \cdot 10^{-11}$ g CO_2e for Optane). However, when using more carbon-intense energy sources like the mix, natural gas, or coal, the operational carbon outweighs the embodied carbon so that Optane becomes more efficient ($8.1 \cdot 10^{-10}$ g CO_2e for DRAM and $2.7 \cdot 10^{-9}$ g CO_2e for Optane when using the mix).

7 OUTLOOK: PROVISIONING SYSTEMS

The components presented with the `carbon-FooBar` approach, namely the carbon model, the collected memory characteristics, and the analysis tools, serve the Memory Governor to make decisions for a system at runtime. However, they can also be used to reason about the carbon footprint of systems ahead of runtime, especially for provisioning new installations. For a specific example: Recent developments in coherent interconnects such as CXL [27] open the design space beyond the choice of memory technologies to entirely new system topologies. With CXL, physical memory transactions that would have been handled by a local memory controller can be forwarded to and handled by remote machines, providing an efficient path for memory disaggregation schemes. Unused memory on one machine can be donated to another machine that would otherwise have exhausted its physical memory capacity, albeit with a latency and bandwidth penalty shown to be slightly larger compared to remote memory accesses in large NUMA systems [25, 28].

Multiple lean machines with low embodied carbon can offset their interconnect overhead compared to a single complex machine. This strongly depends on the workload, as high interconnect traffic between machines may cause a disproportionately large operational carbon footprint. However, for a different workload, allocations may rarely transition between the cache subsystem and physical memory, so both performance losses and communication energy costs will be low. In all three situations, the `carbon-FooBar` workload behaviour model combined with the carbon model can help to anticipate the actual trade-off between embodied carbon savings and operational costs. Additionally, it supports provisioning decisions, complementing previous solutions [1], with performance predictions to ensure operational objectives can be met.

8 RELATED WORK

Research has shown the importance of incorporating system software for *effective* system-wide *energy management* [11, 35]. At the same time, advances in operating systems established support for new hardware properties such as non-volatile, persistent memory [4] in embedded systems [9], data-centre systems [7], and large-scale main-memory database systems [23, 24] as well as disaggregated memory systems [36]. The idea of carbon-aware workload placement, based on application SLAs has been discussed before [3], but with a stronger focus on compute jobs in a cluster compared to the proposed `carbon-FooBar` approach with memory placements within one system. In addition, our position paper also includes the impact of endurance and embodied carbon, which allows for other resources to be used, depending on the excess low-carbon energy supply [31] and the production conditions [18]. However, the combination and joint use of different memory technologies in a single, composable system requires additional support. Special-purpose solutions have been explored in individual cases [33, 34] but generic approaches at the operating system level are still missing. To make efficient use of different types of memory, the operating system needs to adapt applications at runtime (i.e. depending on memory access patterns) to the available hardware resources (i.e. type and size). The idea of scoped allocators to denote requirements for specific buffers has been demonstrated previously in the context of NUMA-aware data placement [14]. Although that work targeted quantitative requirements like latency and throughput, it did so by specifying the desired NUMA node (explicitly or implicitly) rather than by weighted resource characteristics.

9 CONCLUSION

The carbon impact, both embodied and operational, is crucial in the design and operation of computer systems. Heterogeneous systems, composed of novel memory interfaces and cell types, require system software and programming models to catch up to the opening gap between existing abstractions and the underlying technologies.

In this paper, we proposed our `carbon-FooBar` approach towards carbon-aware memory placements with novel memory technologies. `carbon-FooBar` builds on the notion of a Memory Governor, an operating system component that combines knowledge of workload behaviour and sensitivity with available hardware characteristics for allocating memory resources energy-efficiently. Our envisioned `energy-FooBar` notation forms a central vehicle for exchanging and persisting this information. We outlined the interactions and information flow between different components in our architecture that help to tune and balance system energy consumption, carbon-intensity, and modelled embodied carbon through efficient memory placement.

REFERENCES

[1] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. 2023. Carbon Explorer: A Holistic Framework for Designing Carbon Aware Datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 118–132. https://doi.org/10.1145/3575693.3575754

[2] Shoaib Akram. 2021. Performance Evaluation of Intel Optane Memory for Managed Workloads. *ACM Transactions on Architecture and Code Optimization* 18, 3, Article 29 (apr 2021), 26 pages. https://doi.org/10.1145/3451342

[3] Thomas Anderson, Adam Belay, Mosharaf Chowdhury, Asaf Cidon, and Irene Zhang. 2022. Treehouse: A case for carbon-aware datacenter software. *Proceedings of HotCarbon 2022: First Workshop on Sustainable Computer System Design and Implementation* (2022).

[4] Katelin Bailey, Luis Ceze, Steven D Gribble, and Henry M Levy. 2011. Operating System Implications of Fast, Cheap, Non-Volatile Memory. In *Proceedings of the 2011 ACM SIGOPS Workshop on Hot Topics in Operating Systems (HotOS'11)*. 1–5.

[5] Writam Banerjee. 2020. Challenges and Applications of Emerging Nonvolatile Memory Devices. *Electronics* 9, 6, Article 1029 (2020).

[6] Rishiraj A Bheda, Jason A Poovey, Jesse G Beu, and Thomas M Conte. 2011. Energy efficient phase change memory based main memory for future high performance systems. In *Proceedings of the 2011 IEEE International Green Computing Conference*. 1–8.

[7] Daniel Bittman, Peter Alvaro, Pankaj Mehra, Darrell DE Long, and Ethan L Miller. 2020. Twizzler: a Data-Centric OS for Non-Volatile Memory. In *Proceedings of the 2020 USENIX Annual Technical Conference (ATC'20)*. 65–80.

[8] Stacey L. Dolan and Garvin A. Heath. 2012. Life Cycle Greenhouse Gas Emissions of Utility-Scale Wind Power. *Journal of Industrial Ecology* 16, s1 (2012), S136–S154. https://doi.org/10.1111/j.1530-9290.2012.00464.x

[9] Christian Eichler, Henriette Hofmeier, Stefan Reif, Timo Hönig, Jörg Nolte, and Wolfgang Schröder-Preikschat. 2021. Neverlast: An NVM-centric operating system for persistent edge systems. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'21)*.

[10] Electricity Maps ApS. 2022. Electricity Maps. Acc. 2023-05-16. https://app.electricitymaps.com/map

[11] Carla Schlatter Ellis. 1999. The case for higher-level power management. In *Proceedings of the 1999 ACM SIGOPS Workshop on Hot Topics in Operating Systems (HotOS'99)*. 162–167.

[12] Green Software Foundation. 2021. Software Carbon Intensity Standard. https://github.com/Green-Software-Foundation/sci/blob/main/Software_Carbon_Intensity/Software_Carbon_Intensity_Specification.md

[13] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: Designing Sustainable Computer Systems with an Architectural Carbon Modeling Tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA '22)*. Association for Computing Machinery, 784–799. https://doi.org/10.1145/3470496.3527408

[14] Wieland Hagen, Max Plauth, Felix Eberhardt, Frank Feinbube, and Andreas Polze. 2016. PGASUS: a framework for C++ application development on NUMA architectures. In *Proceedings of the Fourth IEEE International Symposium on Computing and Networking (CANDAR'16)*. 368–374. https://doi.org/10.1109/CANDAR.2016.0071

[15] Benedict Herzog, Stefan Reif, Julian Preis, Wolfgang Schröder-Preikschat, and Timo Hönig. 2021. The Price of Meltdown and Spectre: Energy Overhead of Mitigations at Operating System Level. In *Proceedings of the 14th European Workshop on Systems Security* (Online, United Kingdom) *(EuroSec '21)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3447852.3458721

[16] Timo Hönig, Benedict Herzog, and Wolfgang Schröder-Preikschat. 2019. Energy-demand Estimation of Embedded Devices using Deep Artificial Neural Networks. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC'19)*. 617–624.

[17] Mark Horowitz. 2014. Computing's Energy Problem (and what can we do about it). In *Digest of Technical Papers-IEEE International Solid-State Circuits Conference*, Vol. 57.

[18] Donald Kline, Nikolas Parshook, Xiaoyu Ge, Erik Brunvand, Rami Melhem, Panos K. Chrysanthis, and Alex K. Jones. 2019. GreenChip: A tool for evaluating holistic sustainability of modern computing systems. *Sustainable Computing: Informatics and Systems* 22 (2019), 322–332. https://doi.org/10.1016/j.suscom.2017.10.001

[19] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu. 2013. Evaluating STT-RAM as an energy-efficient main memory alternative. In *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'13)*. 256–267. https://doi.org/10.1109/ISPASS.2013.6557176

[20] Edgar A. León, Brice Goglin, and Andres Rubio Proaño. 2019. M&MMs: Navigating Complex Memory Spaces with Hwloc. In *Proceedings of the ACM/IEEE International Symposium on Memory Systems (MEMSYS'19)*. 149–155.

[21] Ingo Molnar, Morten Rasmussen, and Quentin Perret. 2019. Energy Aware Scheduling. Acc. 2023-02-03. https://www.kernel.org/doc/html/latest/scheduler/sched-energy.html

[22] Patrick R. O'Donoughue, Garvin A. Heath, Stacey L. Dolan, and Martin Vorum. 2014. Life Cycle Greenhouse Gas Emissions of Electricity Generated from Conventionally Produced Natural Gas. *Journal of Industrial Ecology* 18, 1 (2014), 125–144. https://doi.org/10.1111/jiec.12084

[23] Ismail Oukid, Daniel Booss, Adrien Lespinasse, Wolfgang Lehner, Thomas Willhalm, and Grégoire Gomes. 2017. Memory management techniques for large-scale persistent-main-memory systems. In *Proceedings of the Very Large Data Base Endowment (VLDB'17)*. 1166–1177.

[24] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. 2016. FPTree: A hybrid SCM-DRAM persistent and concurrent B-tree for storage class memory. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*. 371–386.

[25] Christian Pinto, Dimitris Syrivelis, Michele Gazzetti, Panos Koutsovasilis, Andrea Reale, Kostas Katrinis, and H Peter Hofstee. 2020. Thymesisflow: A software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 868–880.

[26] Stefan Reif, Benedict Herzog, Judith Hemp, Timo Hönig, and Wolfgang Schröder-Preikschat. 2020. Precious: Resource-Demand Estimation for Embedded Neural Network Accelerators. In *Proceedings of the 1st International Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware (CHALLENGE'20)*. mlsys.org, 1–9.

[27] Debendra Das Sharma. 2022. Compute Express Link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*. 5–12. https://doi.org/10.1109/HOTI55740.2022.00017

[28] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Ipoom Jeong, Ren Wang, and Nam Sung Kim. 2023. Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices. *arXiv preprint arXiv:2303.15375* (2023).

[29] Swamit Tannu and Prashant J. Nair. 2022. The Dirty Secret of SSDs: Embodied Carbon. *Proceedings of HotCarbon 2022: First Workshop on Sustainable Computer System Design and Implementation* (2022).

[30] thinkstep AG. 2019. Life Cycle Assessment of Dell Latitude 7300 25th Anniversary Edition. Acc. 2023-05-19. https://www.delltechnologies.com/asset/en-us/products/laptops-and-2-in-1s/technical-support/full-lca-latitude7300-anniversary-edition.pdf

[31] Amanda Tomlinson and George Porter. 2022. Something Old, Something New: Extending the Life of CPUs in Datacenters. *Proceedings of HotCarbon 2022: First Workshop on Sustainable Computer System Design and Implementation* (2022).

[32] Michael Whitaker, Garvin A. Heath, Patrick O'Donoughue, and Martin Vorum. 2013. Second corrigendum to: Whitaker, M., G. A. Heath, P. O'Donoughue, and M. Vorum. 2012. Life cycle greenhouse gas emissions of coal-fired electricity generation: Systematic review and harmonization. Journal of Industrial Ecology 16(S1): S53–S72. *Journal of Industrial Ecology* 17, 5 (2013), 789–792. https://doi.org/10.1111/jiec.12060

[33] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. 2017. HiKV: A hybrid index key-value store for DRAM-NVM memory systems. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC'17)*. 349–362.

[34] Jian Xu and Steven Swanson. 2016. NOVA A log-structured file system for hybrid volatile/non-volatile main memories. In *Proceedings of the 2016 USENIX Conference on File and Storage Technologies (FAST'16)*. 323–338.

[35] Heng Zeng, Carla S Ellis, Alvin R Lebeck, and Amin Vahdat. 2002. ECOSystem: managing energy as a first class operating system resource. In *Proceedings of the 2002 ACM SIGPLAN International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'02)*. 123–132.

[36] Pengfei Zuo, Jiazhao Sun, Liu Yang, Shuangwu Zhang, and Yu Hua. 2021. One-sided RDMA-Conscious Extendible Hashing for Disaggregated Memory. In *Proceedings of the 2021 USENIX Annual Technical Conference (ATC'21)*. 15–29.