Energy Time Fairness: Balancing Fair Allocation of Energy and Time for GPU Workloads

Qianlin Liang
University of Massachusetts Amherst
Amherst, MA, USA
qliang@cs.umass.edu

Walid A. Hanafy
University of Massachusetts Amherst
Amherst, MA, USA
whanafy@cs.umass.edu

Noman Bashir
University of Massachusetts Amherst
Amherst, MA, USA
nbashir@umass.edu

David Irwin
University of Massachusetts Amherst
Amherst, MA, USA
irwin@ecs.umass.edu

Prashant Shenoy
University of Massachusetts Amherst
Amherst, MA, USA
shenoy@cs.umass.edu

ABSTRACT

Traditionally, multi-tenant cloud and edge platforms use fair-share schedulers to fairly multiplex resources across applications. These schedulers ensure applications receive processing time proportional to a configurable share of the total time. Unfortunately, enforcing time-fairness across applications often violates energy-fairness, such that some applications consume more than their fair share of energy. This occurs because applications either do not fully utilize their resources or operate at a reduced frequency/voltage during their time-slice. The problem is particularly acute for machine learning (ML) applications using GPUs, where model size largely dictates utilization and energy usage. Enforcing energy-fairness is also important since energy is a costly and limited resource. For example, in cloud platforms, energy dominates operating costs and is limited by the power delivery infrastructure, while in edge platforms, energy is often scarce and limited by energy harvesting and battery constraints.

To address the problem, we define the notion of *Energy-Time Fairness (*ETF*)*, which enables a configurable tradeoff between energy and time fairness, and then design a scheduler that enforces it. We show that ETF satisfies many well-accepted fairness properties. ETF and the new tradeoff it offers are important, as some applications, especially ML models, are time/latency-sensitive and others are energy-sensitive. Thus, while enforcing pure energy-fairness starves time/latency-sensitive applications (of time) and enforcing pure time-fairness starves energy-sensitive applications (of energy), ETF is able to mind the gap between the two. We implement an ETF scheduler, and show that it improves fairness by up to 2×, incentivizes energy efficiency, and exposes a configurable knob to operate between energy- and time-fairness.

CCS CONCEPTS

• **General and reference → Cross-computing tools and techniques**.

KEYWORDS

FairShare, Energy-aware, Energy-efficiency, Scheduling, Resource Management

ACM Reference Format:
Qianlin Liang, Walid A. Hanafy, Noman Bashir, David Irwin, and Prashant Shenoy. 2023. Energy Time Fairness: Balancing Fair Allocation of Energy and Time for GPU Workloads. In *The Eighth ACM/IEEE Symposium on Edge Computing (SEC '23), December 6–9, 2023, Wilmington, DE, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3583740.3628435

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SEC '23, December 6–9, 2023, Wilmington, DE, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0123-8/23/12…$15.00
https://doi.org/10.1145/3583740.3628435

1 INTRODUCTION

The growth of cloud and edge platforms over the past decade has dramatically lowered the cost of large-scale computing and storage, which has enabled a wide range of cloud- and edge-based applications, including web applications, batch processing, machine learning, and mobile augmented reality. A key to lowering cost has been amortizing it across many diverse users and applications by highly multiplexing resources, which increases both resource utilization and efficiency. Such resource sharing is especially important for lowering the cost of scarce, and thus high-cost, resources, such as GPUs which remain in high demand due to rapidly-growing AI workloads. Consider an edge platform use case as an example, small autonomous vehicles (e.g., food or package delivery robots) may be equipped with an array of sensors, such as cameras, LiDAR, and infrared and perform various tasks including obstacle detection, traffic monitoring, and face detection for pedestrians. Such tasks share the onboard computing and energy systems. However, multiplexing resources introduces a new set of challenges. Operators are required to balance shared resources across diverse tenant applications. Such challenges are inherently present in numerous edge scenarios, especially when constrained by limited resources, inclusive of computational capacity and energy.

Traditionally, multi-tenant cloud and edge platforms have used fair-share schedulers to fairly multiplex server resources across multiple applications. Fair-share schedulers ensure all co-resident applications receive processing time that is proportional to a configurable share of the total processing time, which prevents applications from either starving or dominating the processing time. Fair-share scheduling has been studied for more than three decades with numerous proposed scheduling policies that vary in their underlying assumptions, e.g., definitions of fairness and resource domain, including Weighted Fair Queuing (WFQ) [12], max-min fairness [26],

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

Start-Time Fair Queueing (SFQ) [16], lottery scheduling [49], and Dominant Resource Fairness (DRF) [15]. Fair-share schedulers are also widely implemented in modern operating systems (OSes) (e.g., Linux [36]), hypervisors (e.g., Xen and VSphere [52]), batch schedulers (e.g., Slurm [2]), container orchestration platforms (e.g., Kubernetes [6], and data-processing platforms (e.g., Spark and Hadoop [1]).

Importantly, the traditional fair-share schedulers above enforce time-fairness across multi-tenant applications such that each application receives processing time in proportion to their allocated share. Unfortunately, enforcing time-fairness across applications often violates energy-fairness, such that the share of energy each application consumes is substantially different from their allocated share of time. This divergence between time- and energy-fairness occurs because applications may either not fully utilize their resources or operate at a reduced frequency/voltage during their allocated time-slice. Enforcing energy-fairness is crucial for both cloud and edge, as energy is a costly and limited resource. In cloud platforms, while energy substantially dominates operating costs, its supply is inherently limited by the power delivery infrastructure. This creates a need for fairness to ensure no single tenant disproportionately consumes this costly resource, potentially leading to inequities in service delivery. On the other hand, edge platforms often grapple with energy scarcity. Here, energy is not just about cost but about availability, often restricted by energy harvesting techniques and battery capacities. Unfair allocation can lead to premature battery depletion in some devices, rendering them nonfunctional and breaking the continuity of services they provide. In addition, energy-efficiency and sustainability are increasingly important in reducing computing's environmental impact [3, 42, 44]. The problem is particularly acute for machine learning (ML) applications using GPUs, where model size largely dictates GPU utilization and energy usage. Similarly, on edge platforms with energy constraints, applications often reduce frequency/voltage during their time-slice to conserve energy.

Enforcing energy-fairness not only directly contributes to a reduced energy consumption and carbon footprint but also serves as a strategic lever, motivating users to refine and enhance the energy-efficiency of their applications (i.e., smaller models and lower frequencies), which is important in both cloud and edge settings. Unfortunately, there is a fundamental conflict between time-fairness and energy-fairness, such that simultaneously enforcing both is not always possible. To see why, consider a system with dynamic voltage and frequency scaling (DVFS), which is widely used by CPUs and GPUs to minimize energy usage by reducing frequency and voltage when demand is low. While OSes traditionally set a single frequency/voltage state for all applications, modern GPUs enable each application to set their own custom frequency/voltage for its time-slice. However, in this case, selecting a lower frequency/voltage state to conserve energy results in an application executing fewer computations during its allocated time-slice, which effectively penalizes it for being more energy-efficient. Since applications operating at lower frequency/voltages consume much less energy than others, their share of energy consumption will be much less than their share of processing time. This problem also manifests itself on GPUs due to differences in model size and complexity. Since most GPUs are only time-shared and not space-shared, they load and run only a single model at any time. GPU energy usage is largely dictated by a model's size and complexity, as this determines the internal GPU resources a model utilizes. In general, smaller models will consume less energy than larger ones during their time-slice.

Prior work has identified the problems above and developed schedulers that enforce energy-fairness across multi-tenant applications instead of time-fairness [13]. Unfortunately, pure energy-fair schedulers have the opposite problem to the one above: by enforcing energy-fairness, they sacrifice time-fairness. That is, an energy-fair scheduler allows applications running at a low frequency/voltage setting (or executing small models) to receive disproportionately large time-slices, which starves applications running at high frequency/voltage (or executing large models). In general, some applications may be time/latency-sensitive, while others may be energy-sensitive. As our examples above show, enforcing pure energy-fairness starves time/latency-sensitive applications (of time) while enforcing pure time-fairness starves energy-sensitive applications (of energy). A key insight of our work is that there is a fundamental dependency between an application's time and energy allocation such that enforcing fairness in one inherently results in unfairness in the other. Since time-fairness and energy-fairness both provide important properties, cloud and edge platforms should enforce a balance between them.

To address the problem, we define a novel notion of Energy-Time Fairness (ETF), which enables a configurable tradeoff between energy and time fairness, and then design a scheduler that enforces it. We show that ETF satisfies many well-accepted and desirable fairness properties, including an energy-efficiency and sharing incentive, strategy-proofness, and pareto efficiency [15]. More generally, while prior work has shown how to enforce fairness when allocating multiple independent resources (e.g., cores and memory) [15], ETF shows how to enforce fairness for multiple dependent resources (e.g., processing time and energy). ETF recognizes that it is impossible to simultaneously enforce both time-fairness and energy-fairness across applications, and thus instead defines a smooth and configurable tradeoff between these two competing dimensions of fairness.

Our hypothesis is that using ETF can bridge the gap between pure time and energy fairness and thus support both i) a configurable incentive for operating energy-efficiently and ii) running time/latency-sensitive applications. In evaluating our hypothesis, we make the following contributions.

Fairness Conflict. To motivate our work, we first experimentally demonstrate the conflict between time-fairness and energy-fairness by showing that i) enforcing time-fairness starves some applications of energy and ii) enforcing energy-fairness starves some applications of time. To our knowledge, this paper is the first to highlight the conflict between time and energy-fairness.

Energy-Time Fairness. We define the notion of Energy-Time Fairness (ETF), which enables a configurable tradeoff between energy and time fairness, and show that it satisfies many well-accepted and desirable fairness properties, including Pareto efficiency and strategy proofness, while also balancing competing dimensions of fairness for dependent resources.

Implementation and Evaluation. We implement ETF in a new scheduling framework, called ETFS, that is designed for edge and cloud GPUs. We demonstrate that ETF improves the fairness by

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

up to 2×, incentivizes energy efficiency, and allows a configurable tradeoff between energy-fair and time-fair scheduling for multi-tenant GPU model serving workloads.

2 BACKGROUND

This section provides background on fair-share scheduling and energy optimization approaches in cloud and edge settings.

2.1 Cloud and Edge Multitenancy

Our work targets typical cloud and edge platforms that serve multiple customers and their applications. Cloud and edge servers in these environments host multiple applications from different customers using virtual machines or containers. Doing so enables multiplexing server resources across multiple applications, increasing a system's utilization and providing isolation across applications. Our work assumes a multi-tenant environment where multiple applications share a server's CPU or GPU resources. We assume that the OS or hypervisor scheduler uses a fair-share scheduling policy as the underlying resource management technique to multiplex server resources across multi-tenant applications.

2.2 Fair-share Scheduling Overview

Fair-share schedulers are a class of scheduling algorithms that enable each application to be allocated a share (or slice) of the underlying resource and enforce these shares when making scheduling decisions. The share is often specified in terms of a weight W, or a fraction of the resources, such that the resource is time-shared among applications in proportion to the weight or fraction. For example, if two applications are given weights 1:1, each application receives 50% of the time slices on the underlying resource. Early fair-share schedulers, such as Weighted Fair Queuing (WFQ) [12] and Generalized Processor Sharing (GPS) [37], applied this idea to network routers and server processors, respectively.

Modern variants, such as Dominant Resource Fairness (DRF) [15] and Dominant Resource Fair Queuing (DRFQ) [14], generalized fair-share scheduling to multi-resource settings based on the concept of a dominant resource. Fair-share schedulers are typically based on an underlying mathematical notion of fairness, such as MAX-MIN fairness [26], that they strive to achieve through their scheduling decisions. A detailed discussion of fair-share schedulers is presented in Section 7.

2.3 Energy Optimization

A complementary consideration to fairly sharing resources is the energy-efficient use of edge and cloud resources. The recent emphasis on the sustainability of cloud data centers has made energy-efficiency techniques an increasingly important consideration. While there exist many approaches for optimizing the energy consumption of cloud and edge applications, we highlight two primary strategies to improve energy efficiency for systems: *algorithmic improvement* and *system-level configuration*. Algorithmic improvements optimize the energy-efficiency and throughput using more efficient algorithms, data structures, and communication paradigms [10]. In contrast, system-level configurations, such as dynamic voltage and frequency scaling (DVFS), increase the efficiency by enabling the system to regulate the CPU or GPU energy footprint, e.g., by adjusting its voltage and/or frequency in the case of DVFS. Lowering the voltage or frequency slows down the execution speed of the application, while also reducing the energy consumption. In particular, the power dissipated by the CPU or GPU is governed by the CMOS chip presented in this well-known equation:

$$P = \alpha C f V^2 \tag{1}$$

where α is a proportional constant indicating the percentage of the system that is active or switching, C represents the system's capacitance, f is the frequency at which the system is switching, and V denotes the voltage swing across C.

DVFS techniques have been studied for nearly two decades and have become commonplace in today's processors and GPUs. A common approach involves having the OS determine a common operating frequency for the entire system, and all applications, based on their current demand. In some systems, schedulers have begun allowing applications to choose their own desired operating frequency. For example, Nvidia's line of GPUs allows individual applications to choose their own frequency setting whenever they execute. When applications choose a custom DVFS setting, rather than a system-wide value, different applications will execute at different power levels during different time slices. This has important implications for fair-sharing, as discussed in section 3.

2.4 Model Serving Workloads

While our current work on energy time fairness has broad applicability to a range of applications, we highlight its efficacy using a particular class of edge workloads, namely model serving, which involves running multiple deep learning models on a shared resource such as a GPU. This workload choice is motivated by the increasing energy demand of AI applications, which have made energy efficiency a key goal in designing edge and cloud systems [33, 35, 50]. Along with the increasing use of application-specific accelerators, such as GPUs, researchers have proposed multiple algorithmic improvements to increase their models' energy efficiency, usually for a minimal degradation in accuracy. Many of these improvements can be viewed as design-time approaches, where users select their DNN architecture and parameters [25, 45, 58]. Separately, researchers have designed accelerator-specific methods, where DNNs are able to gain much higher energy efficiency by customizing models to specific hardware [5, 22, 57]. Researchers have also proposed post-training methods that alter the architecture and parameters after training is complete in a static fashion, such as pruning [21, 54], distillation [24] and quantization [23], or by allowing the network to alter its resource requirements in run-time [8, 46, 56]. Finally, we highlight that resource sharing in model-serving systems is a key objective, as shown in previous research [11, 18, 20, 32, 33, 40].

3 MOTIVATION FOR ENERGY-TIME FAIRNESS

This section discusses some key assumptions made by traditional fair schedulers and why the assumptions do not hold for emerging hardware, such as modern GPUs and heterogeneous processors, resulting in unfairness problems.

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

3.1 Fair-Share Scheduling Basics

Classical fair share schedulers make three implicit assumptions that are starting to break down with emerging CPU and GPU architectures in edge and cloud servers. First, since multi-tenant systems are time-shared across resident applications, it is natural for fair-share schedulers to allocate CPU or GPU resources at the granularity of time slices. For instance, a fair-share scheduler can achieve an allocation in the ratio 1:2 by allocating one time slice to application `A` and two time slices to application `B` in a round-robin manner. Second, fair-share schedulers implicitly assume that applications execute at the same speed (e.g., same clock frequency) across time slices, which implies that they are allocated the same number of CPU or GPU cycles in each allocated time slice.

The fairness guarantees of such schedulers are predicated on all applications making equal progress on a weighted basis, which is achieved by allocating equal, or proportional, amounts of CPU or GPU cycles. At equal execution speeds, allocating an appropriate fraction of time slices equates to allocating an appropriate fraction of the CPU or GPU cycles. Third, fair-share schedulers implicitly assume the energy used by an application is proportional to their execution time. Thus, if two applications execute for one time slice each at the same execution speed (i.e., same CPU or GPU frequency), the scheduler assumes they will consume the same amount of energy in their corresponding time slices. This assumption generally holds since power consumption depends directly on the operating clock frequency of the CPU or GPU.

While these implicit assumptions have largely held for processors for many decades, the emergence of heterogeneous core processors and multi-core GPUs are causing them to break down in many scenarios. The wide availability of DVFS technology on modern CPU and GPUs, combined with the ability for applications to run at different frequencies, means that it is no longer logical to assume that applications will consume identical cycles and power during their time slice [19, 51]. The operating frequency may be changed frequently either by the OS or, in some cases, by the applications itself. Second, the rise of heterogeneous multi-core processors in modern CPUs and GPUs means that equal time execution no longer implies equal energy consumption [27, 34, 47]. This assumption already breaks down under DVFS, since applications executing at different frequencies will consume a different amount of power on homogeneous cores. In heterogeneous settings and for data-parallel GPU cores, even executing at the *same frequency* will yield different amounts of progress and *consume different amounts of energy*. These trends have important implications for fair-share schedulers at edge.

3.2 Unfairness in Classical Fair-share Schedulers

Given the observations above, we present three different cases of unfairness in traditional fair-share schedulers when cloud and edge servers also employ energy optimizations.

3.2.1 Unfairness due to application heterogeneity. Consider two applications, each consisting of a deep learning model, that share a GPU for performing inference tasks. This multi-tenant application is a common case for edge computing where IoT devices send data to edge servers that perform DNN inference over their data [32, 33]. Multi-tenant applications are handled using a *model-serving* paradigm, where the server hosts and services multiple DNN models on its GPU resources. Since the GPU is time-shared, we assume the scheduler allocates time slices to execute each DNN model and its inference requests using fair-share scheduling. Notably, due to the differences in DNN model size and architecture, even when both applications use the same GPU frequency and number of time slices, they can consume different amounts of energy.

Figure 1: ***Power consumption of different ML models when run under different frequencies on an Nvidia TX2.***

(a) Time Fair **(b) Energy Fair**

Figure 2: ***Unfairness in time and energy-based fair schedulers.***

To illustrate, Figure 1 shows the power consumption of serving 4 different models at various GPU frequencies when running using Nvidia TX2. The figure shows that each model will consume different amounts of energy within its time slice. The primary cause of these differences is that GPUs consist of many processing units, e.g, CUDA and Tensor cores, to satisfy a diverse set of application requirements of data parallelism and models with different sizes. Hence, different models exercise different fractions and parts of the GPU cores. Since the models have different architectural structures and sizes, where highly parallel models can utilize more cores of the GPU than thin models, they will incur different amounts of energy use, as highlighted in recent research [22, 29, 35]. Importantly, fair-share schedulers try to equalize application progress on a weighted basis, with some schedulers even allocating resources in units of cycles, rather than time, to achieve proportionate progress. This can cause smaller models that use fewer cycles (and GPU cores) to receive proportionately larger amounts of GPU time and possibly starve larger DNN models. That is, larger DNN models that try to utilize all available cores effectively get penalized and may starve. This is clearly undesirable since resource-efficient use should be favored rather than penalized.

3.2.2 Unfairness of Time-based fair Schedulers. Next, consider a conventional time-based fair-share scheduler that allocates time

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

slices in proportion to assigned weights. Consider two GPU applications that are assigned equal weights. Since GPUs, such as those from Nvidia, permit fine-grain DVFS settings, applications can choose their own DFS frequency. Assume two applications `A` and `B`, where application `A` runs a `Resnet50` model at 1300 MHz GPU core frequency and application `B` runs a `MobilenetV2` model at 726 MHz GPU core frequency on an Nvidia TX2. Figure 2a depicts the time allocated to each application as well as their energy usage. As shown, the resource allocation is fair in the time dimension, since each application receives an equal number of time slices. However, the application executing at full speed consumes more energy than the one running at a lower speed, which implicitly favors the former application by allowing it to use a larger fraction of the system's energy despite being allocated the same weight. Thus, the allocation is unfair for energy.

3.2.3 Unfairness of Energy-based fair Schedulers. The notion of energy fair-share scheduling has been proposed to allocate resources fairly in units of energy, rather than time [13, 19, 51]. In this case, two applications will be scheduled on the resource, such that their energy use is in proportion to their weights. Figure 2b shows the same applications as the previous time fair-share scenario but under energy fair-share scheduling of the GPU. Figure 2b shows that the two applications now use equal amounts of energy. However, to do so, the application running at a lower frequency is allocated much more GPU time to equalize its energy use with respect to the second application. Importantly, the lower the GPU frequency (and power) used by the first application, the more GPU time it receives, possibly starving applications that run at higher frequencies. This results in unfairness in the time dimension.

3.3 Desirable Fairness Properties

Our three motivating examples show that time, energy, and cycles are all tightly coupled with each other. In cloud and edge servers with heterogeneous applications and heterogeneous multi-core hardware, conventional fair schedulers can become unfair in one or more dimensions, and it is *infeasible to simultaneously be fair in all dimensions*. Consequently, we need a new notion of fairness such that edge and cloud systems can achieve their energy-efficiency goals while also being fair to their multi-tenant workloads.

To address the unfairness disparity, we propose a new notion of fairness called Energy-Time Fairness (`ETF`) that bridges the gap between traditional fairness techniques. Our design stems from the unfavorable effects of traditional fairness techniques while guaranteeing their key properties such as:

(1) *Pareto-efficiency*, which ensures high utilization by ensuring that applications cannot increase their resource shares without decreasing others;
(2) *Strategy-proofness*, which ensures that applications don't benefit from lying about their resource demand;
(3) *Sharing incentives*, which promotes sharing the resources dynamically instead of static allocations.

In addition, `ETF` extends the notion of fairness by also providing these additional desirable properties:

(1) *Efficiency incentives*, which reward energy-efficient applications with more time on the resource;
(2) *Time guarantees*, where applications are guaranteed a time slice irrespective of their energy efficiency;
(3) *MAX-MIN energy fairness*, which maximizes the minimum energy allocation among applications.

4 ENERGY-TIME FAIRNESS

We present Energy-Time Fairness (`ETF`), a new concept of fairness that provides minimum time guarantees to prevent starvation, while targeting energy fairness by maximizing the minimum energy across applications with heterogeneous rates of energy consumption in multi-tenant scenarios.

`ETF` allows a system operator to allocate a configurable portion of the scheduling quantum, T, in a time-fair manner to ensure that even the resource-hungry applications that starve under pure energy fair scenario get a minimum guaranteed time slice. However, the ultimate goal of the approach is to be energy fair, so it adjusts the rates for the remaining segment of the time quantum to achieve energy fairness.

4.1 Defining Energy-Time Fairness

Assume there are N applications denoted by A and indexed by i, where application A_i uses power P_i and has weight w_i.

Energy Fairness (EF). The time allocation of the application A_i under a perfect EF scenario, T_i^{EF}, over a scheduling horizon of time duration T is computed as,

$$T_i^{EF} = (1 - \frac{w_i \times P_i}{\sum_{k=0}^{N} w_k \times P_k}) \cdot T. \tag{2}$$

The energy consumption E_i^{EF} of a given application A_i is then computed as,

$$E_i^{EF} = P_i \times T_i^{EF}.$$

Finally, the energy consumption normalized by weight of all applications under EF, should be the same, as shown below.

$$E_i^{EF}/w_i = E_j^{EF}/w_j, \quad \forall\ i, j \in [[1, N]]. \tag{3}$$

Time Fairness (TF). The energy consumption, E_i, under a purely time fair share (TF) scenario can be computed as,

$$E_i^{TF} = P_i \times \frac{w_i}{\sum_{k=0}^{N} w_k} \times T. \tag{4}$$

The time allocation for an application under time fair share (TF) scenario is,

$$T_i^{TF} = \frac{w_i}{\sum_{k=0}^{N} w_k} \times T. \tag{5}$$

The energy consumption across applications differs under TF and deviates from the perfect energy fair-share allocation. The difference between the two can be computed as,

$$\Delta E_i^{unfair} = E_i^{TF} - E_i^{EF}. \tag{6}$$

Energy-Time Fairness (`ETF`). As motivated in the last section, we want to provide minimum time guarantees to prevent starvation of resource-hungry applications while trying to achieve energy fairness by maximizing the minimum energy consumption across applications. To do so, we introduce the notion of *time-fair factor*, denoted by ϕ, which configures the minimum time guarantees to

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

each application as a fraction of time allocated under TF scenario. It is computed as,

$$\min T_i^{ETF} = \phi \times T_i^{TF}. \tag{7}$$

During this time period, the energy consumption of an application A_i is expressed as,

$$\min E_i^{ETF} = \min T_i^{ETF} \times P_i \tag{8}$$

The total time allocated to provide minimum time guarantees is $\phi \times T$. Our approach uses rest of the time, $(1-\phi) \times T$ to improve energy fairness. Our approach targets that improvement using a two step process. First, we allocate the remainder of the time based on EF, which is computed as,

$$\Delta' T_i^{ETF} = (1-\phi) \times T_i^{EF}. \tag{9}$$

While this ensures that the remainder of the time slot is allocated in an energy fair share manner, it does not *correct* the unfairness that was caused while providing minimum time guarantees. The extent of energy unfairness depends on the value of ϕ and the power difference across applications. If an application runs under TF for $\phi \times T$ duration, the deviation of its energy consumption from fair energy fraction can be computed as,

$$\Delta E_{\phi,i}^{unfair} = \phi \times \Delta E_i^{unfair}. \tag{10}$$

Prior work on fair allocations has explored keeping track of *unfairness* and allowing disadvantaged applications to *catch up* later [4, 48]. Inspired by this work, we enable applications to catch up by updating $\Delta' T_i^{ETF}$ proportional to the unfairness to mitigate existing unfairness. The updated rate is computed as,

$$\Delta T_i^{ETF} = \Delta' T_i^{ETF} - \frac{\Delta E_{\phi,i}^{unfair}}{\sum_{k=0}^{N} \Delta E_{\phi,k}^{unfair}}. \tag{11}$$

If an application's $\Delta E_{\phi,i}^{unfair}$ is positive, it means that the application disproportionally used more energy than EF and its rate would further decrease and vice versa.

The total time allocated to an application is a function of ϕ, weights, and power consumption of all the applications. It can be computed as,

$$T_i^{ETF} = \min T_i^{ETF} + \Delta T_i^{ETF}. \tag{12}$$

Similarly, the total energy consumed by the applications is expressed as,

$$E_i^{ETF} = P_i \times T_i^{ETF}. \tag{13}$$

The energy repatriated during the remainder of the catch up period is the difference between the energy fair assignment and new allocation. It can be computed as,

$$\Delta E_{\phi,i}^{repat} = \phi \times E_i^{EF} - \Delta E_i^{ETF}. \tag{14}$$

Achieving Energy Fairness with ETF. At the extreme values of ϕ, ETF devolves to either energy fairness ($\phi = 0$) or time fairness ($\phi = 1$). As we increase the value of ϕ, our approach deviates from energy fairness. If the value of ϕ increases beyond a certain value, the energy unfairness increases to an extent that it cannot catch up in the remainder of the time. We denote this value as ϕ_{fair} and at this point, the energy unfairness caused during the time-fair period is equal to the energy repatriated during the catch up period,

$$\Delta E_{\phi,i}^{unfair} = \Delta E_{\phi,i}^{repat}. \tag{15}$$

This equation can be solved to compute the value of ϕ_{fair}. For values between 0 and ϕ_{fair}, our approach is able to achieve perfect energy fairness while providing the minimum time guarantees given by Eq. 7.

Finally, it is worth noting that the ETF model can effectively be utilized to guarantee a minimum energy share, while also prioritizing the equitable distribution of time.

4.2 ETF Scheduling Algorithm

We consider the following set up for the problem: we have N active applications time-sharing a single resource (e.g. GPUs), which could consumes energy at varying rates, depending on how applications utilize it during their respective usage periods. A time quantum consisting of T time units is allocated amongst them. Each application A_i is allocated a time slice denoted as T_i, such that the sum of all time slices for all applications does not exceed T. Each application then executes their applications within their allocated time slices, operating at a power level denoted as P_i. The energy consumption of each application is therefore not only a function of the time for which it runs but also the power at which it operates. ETF achieves the design objectives by incrementally allocating time slices to each application based on their respective demand, weight, and power usage. This process follows two primary steps: 1) guaranteeing minimum time fairness, and 2) prioritizing energy efficiency.

Guaranteeing minimum time fairness. In this step, ETF addresses the objective of limiting performance degradation by ensuring that no application receives a time slice that falls below a ϕ-fair time share allocation as shown in equation 7. This minimum share is set to ensure that even in scenarios where energy efficiency is being prioritized, the performance of each application is not unduly compromised. This step is depicted in line 1 of algorithm 1. Notably, when an application's demand is lower than the minimum time bound, ETF would allocate only the amount equal to the application's demand. This approach ensures that applications allocations never surpass their demands, which is a necessary condition for achieving Pareto efficiency.

Prioritizing energy efficiency. In the previous step, we distributed $\phi \cdot T$ time units evenly among all applications. However, such an equal distribution of time units can lead to significant disparities in energy consumption. To encourage energy awareness applications are repatriated the amount of energy they lost in the first round by extending their time allocation as explained in equations 12 and 13. Since time slices are indivisible, we use an iterative approach to distribute the remaining time slices. Lines 2 to 9 in algorithm 1 explains these steps. First (step 2) we calculate the current energy consumption based on their power profiles and the time slices that have been allocated to them so far as described in equation 8. Next, we compute the remaining time slices that still need to be allocated (line 3). Subsequently, we start allocating the remaining time slices individually to the application with the minimum current energy consumption (line 5-6). This process continues until the remaining time slices have been allocated or all applications' demands have

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Compute time slice

Input: `weights` w_i, weights of application i.
`demands` d_i, demands for application i.
`power` P_i, power usage for application i.
N, number of active application.
T, schedule interval.
$\phi \in [0, 1]$, guarantee fraction of fair time share.

Output: `timeSlice[`i`]` time slice for each application i.

```
timeSlice[i] = min( weights[i] / (Σ_{k=0}^{n-1} weights[k]) · T · φ, demands[i])
energy[i] = timeSlice[i] · power[i]/weights[i]
remainSlice = T − Σ_{k=0}^{n-1} timeSlice[k]
while remainSlice > 0 and at least one demand is not met do
    k = arg min_i energy[i] if timeSlice[i] < demands[i]
    timeSlice[k] += 1
    energy[k] = timeSlice[k] · power[k]/weights[k]
    remainSlice -= 1
end
```

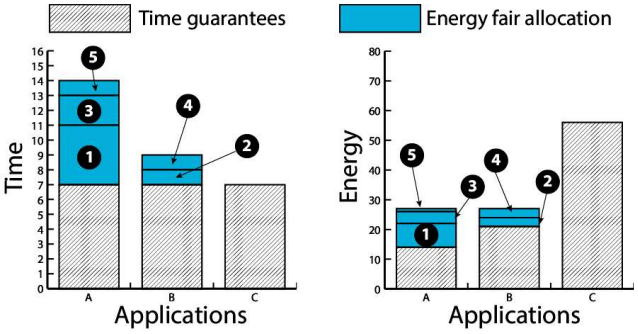

Figure 3: ***An illustrative example of*** `ETF` ***at work.***

been fulfilled. Similar to the previous step, we ensure that we do not allocate time slices to applications whose demands have already been met.

4.3 An Illustrative Example

We now walk through a concrete example. Consider a system with time quantum size of $T = 30$ serving three applications: `A`, `B` and `C`. For simplicity, we assume that these applications have equal weights and are backlogged. Applications `A`, `B`, and `C` have power demands of 2, 3, and 8 watts, respectively. Figure 3 shows time and energy allocations across the applications scheduled under `ETF` at $\phi = 0.7$.

In the first step, `ETF` distributes $\phi \cdot T = 21$ time units evenly among the applications. Since the applications' weights are equal and they are backlogged, each application receives 7 time units. Consequently, their respective energy consumption are 14, 21, and 56 for applications `A`, `B`, and `C`, respectively. The textured bars in figure 3 show the allocations of this step.

Next, `ETF` distributes the remaining $(1 - \phi)T = 9$ time units by prioritizing the applications consuming less energy. Since `A` currently has the least energy consumption, `ETF` allocates additional time units to application A until its energy consumption *catches up* with others. In this round, indicated by the number ❶ in figure 3, `ETF` allocates 4 additional time units to `A`, to be assigned equal energy as `B`. After this round, the time slices for `A`, `B`, and `C` are 11, 7, and 7 and the energy consumption are 22, 21, and 56 respectively.

Now, `B` becomes the application with the least energy consumption. Consequently, `ETF` allocates an additional time unit to application `B` in round ❷. This process continues iteratively, until all of the 30 time units have been allocated after round ❺. After the final allocation, the time slices assigned to applications `A`, `B`, and `C` are 14, 9, and 7, while the allocated energy are 28, 27, and 56, respectively.

4.4 `ETF` Properties

In this section, we prove that `ETF` achieves a set of desirable properties explained in section 3.3, including Pareto efficiency, strategy-proofness and maximizing minimum energy consumption with minimum time bounds.

Theorem 1 (Pareto efficiency). *When the workloads are backlogged, it is not possible to increase the allocation of an application without decreasing the allocation of at least one other application.*

`ETF` is Pareto efficient by design. First, it ensures that no application is allocated more time than their demand. Second, it exhaustively allocates all available time units. Hence, there is no other allocation that can provide more time to an application without taking away time from another application.

Theorem 2 (Strategy-proofness). *In* `ETF`*, applications cannot gain a more useful allocation by misrepresenting their energy and time demands.*

To prove this theorem, we first prove the following Lemmas:

Lemma 1. *Energy fair allocation is strategy-proof.*

Proof. We prove this lemma by contradiction. Let $\langle t_i, e_i \rangle$ be the time and energy allocation for application i using energy fair allocation (equation 2). Since energy fair allocation maximizes the minimum energy consumption among applications (the proof will be provided later in this section), it results in equal energy consumption, see equation 3.

Let's assume that there exists an alternative allocation $\langle t'_i, e'_i \rangle$ such that either $t'_i > t_i, e'_i \geq e_i$ or $t'_i \geq t_i, e'_i > e_i$. If $e'_i > e_i$, it would result in $e'_i > e_j$ for $i \neq j$, which contradicts the condition that the energy consumption is equal across all applications under energy fair scheduling. If $t'_i > t_i$, it would decrease the time received by other applications due to the Pareto efficiency property, and consequently, the energy received by other applications would also decrease. This would lead to $e'_i < e_i$, contradicting our initial assumption.

Since these contradictions arise from our assumption, we conclude that such an allocation cannot exist. Therefore, this demonstrates that energy fair allocation is strategy-proof. □

Lemma 2. *In the absence of knowledge of other application's energy demands, becoming either energy-efficient or energy-gluttonous does not yield an advantage over other tenants.*

This behavior results from the minimum time scheduling (step 1) of `ETF`, where a minimum amount of time is allocated to each application regardless of their energy consumption. If an application

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

is so energy-gluttonous that they receive no time units in step 2, they can benefit by increasing their energy consumption further. Since they have already received the minimum time allocation, increasing their energy consumption does not result in any time penalty. For example, consider the scenario depicted in figure 3: if application C were to increase the power profile from 8 to 10, the energy application C ultimately receives would increase from 56 to 70 without any decrease in time allocation.

Moreover, an application that consumes so little energy that all remaining time units in step 2 are allocated to them (i.e., despite the additional time allotted, they still consume the least amount of energy) can increase the energy consumption to match that of the second least energy-consuming application without incurring any time penalty.

However, both these scenarios are highly contingent on the energy profiles of other applications. Without knowledge of other applications' energy consumption, an application cannot accurately determine its scenario, making it uncertain whether the potential gain would be realized. Thus, manipulating time and energy demand can lead to decreased useful allocations. This effectively deters applications from exploiting the system for additional allocation, thereby enhancing the strategy-proofness of ETF.

Intuitively, ETF maximizes the minimum energy consumption by allocating time units to least energy-consuming applications. We prove it using the following lemma:

LEMMA 3. *Let e_{min} denote the minimum energy consumption among all applications after iteration t of energy fair allocation. For any application i that receives a time allocation during the first t iterations, removing one unit from an application i will result in the energy consumption $e_i \leq e_{min}$.*

PROOF. We prove by contradiction. Assume that $e_i > e_{min}$ after removing one time unit from application i. This indicates that when allocating the removed time unit, the application i is not the least energy-consuming application, and the allocation of the time unit to application i contradicts ETF. □

THEOREM 3. ETF *maximizes the minimum energy consumption among applications while providing the minimum time guarantee.*

PROOF. The minimum time guarantee follows trivially by construction. Suppose application i has the least energy consumption after allocation, with $\langle t_i, e_i \rangle$ being the time and energy allocation for application i. Assume there exists an alternative allocation where the minimum energy allocation is higher, and application i has an allocation of $\langle t_i', e_i' \rangle$ in this case. Note that in this scenario, we must have $e_i' > e_i$ and thus $t_i' > t_i$.

Since application i is receiving more time in this new scenario, due to the Pareto efficiency property, the system must decrease the time allocation for at least one other application, say application j, to maintain the same total time. However, by Lemma 3, the reduction of time from application j will result in the energy consumption for application j, $e_j \leq e_i$, which contradicts our assumption that the minimum energy consumption is higher in the new scenario. Hence, ETF does maximize the minimum energy consumption among applications, as desired. □

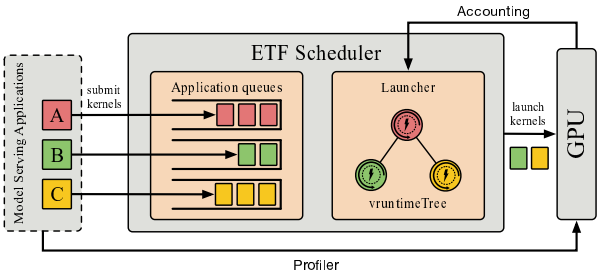

Figure 4: *Architecture of* ETF *implementation.*

5 ENERGY TIME FAIR SCHEDULER IMPLEMENTATION

Although the concept of ETF can be used to arbitrate any shared resource, we have decided to implement it as a GPU scheduler due to its important role in many cloud and edge systems. The structure of the scheduler is depicted in figure 4. Our scheduler is a user-space scheduling library that abstracts the GPU and provides application-level interfaces similar to what the GPU runtimes, such as CUDA, offers[1]. The only difference is that the scheduling decisions incorporate the notion of Energy Time Fairness rather than time fairness alone. In this system, applications submit their CUDA kernels to their respective queues, similar to how applications submit their requests to CUDA streams. Upon receipt of these kernels, the system calculates the application's share, monitors their resource consumption, and determines the sequence of kernel execution. Following this, the scheduler launches the next kernel to the GPU or the CPU processor, which is shared among multiple applications, with tasks from different applications interleaved over time. The scheduler is implemented in C++ using about 2,000 lines of code, and will be open-source at `github.com/umassos/energy-time-fairness`.

In the rest of this section, we discuss the details of the scheduler, the utilized model-serving application, and the application profiler.

Scheduler. In designing the ETF scheduler, we had two key objectives in mind: *compatibility* and *efficiency*. The scheduler should be compatible with current frameworks APIs and the scheduling decisions overhead should be minimal. To achieve the first goal our scheduler API closely mimics that of GPU runtimes such as CUDA. In this case, when the application submits a kernel to the GPU our ETF scheduler queues this request and submit it later based on the Energy Time Fair resource allocation. Our application queue applies the same first come, first served (FCFS) as CUDA streams to guarantee correctness. It should be noted that a single application may have multiple queues similar to how applications can have multiple CUDA streams.

The second objective was to minimize the overhead of the scheduler. The scheduling overhead stems from three sources, the application queues, resource accounting, and application ordering decisions. To facilitate an efficient queuing process, the application queues are implemented using doubly-linked list data structures, which allows us to support enqueue and dequeue operations in $O(1)$ time. ETFS counts time using the virtual runtime (`vruntime`) technique similar to the Completely Fair Scheduler (CFS). As each

[1]The implementation can be extended to intercept kernel calls, which don't require application modifications

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

kernel executes, it accumulates vruntime, defined as:

$$\texttt{vruntime}_i = \texttt{vruntime}_i + \frac{t_i}{timeSlice_i} \tag{16}$$

Where, t_i is the actual runtime of application i's time and $timeSlice_i$ is the time slice computed by Alg.1 using the time and power consumption profiles collected at run time or through a profiler. The scheduler also accounts for energy consumption to ensure that applications respect their share.

When a scheduling decision occurs, the scheduler selects the application, denoted by j, with the lowest vruntime to execute next. Specifically, the scheduler pops the task at the head of application j's queue and allows it to run for $timeSlice_j$ time units or until the application's queue becomes empty. Note that the scheduler only considers applications who have non-empty queues, meaning it will not assign time slices to applications with no incoming requests. To achieve efficient ordering, we implement vruntime counting using a Red-Black Tree data structure. Applications are ordered in the tree by their vruntime, allowing for most operations such as insertion and deletion to be executed in logarithmic time (i.e. $O(\log n)$).

Model-Serving applications. To evaluate the ETF scheduler we used model-serving applications. In model-serving systems, each request is processed by submitting a batch of kernels that execute sequentially and the result is only available when the last one finishes. Since our scheduler is a user-space library that required altering the way kernels are submitted to the GPU, we utilized Apache TVM[9], an open-source machine learning compiler framework, to utilize our scheduling library to submit kernels to the ETF's queues rather than directly to the GPU.

Application Profiling. When using the ETF approach, it is important to keep track of time and energy consumption in a detailed manner. While it is possible to do this in a way that is not specific to the application being used, we have found that this is not always accurate, particularly when it comes to energy usage. This is because GPU kernels typically only last for a few milliseconds or less, and even with the presence of built in power monitoring, the propagation delay is much higher. To overcome this issue, we perform a one-time profiling each time a new model is loaded. This process gathers information about the resources consumed by the DNN during task execution. The profiler achieves this by repeatedly executing the inference requests for 5 seconds using the model and collecting the following profile data: (i) average kernels service time, which indicates the amount of time required to process the kernel; (ii) average power usage when executing the kernels. To ensure accurate profiling, we monitor the variance of measurements. If the coefficient of variance exceeds 5%, we redo the profiling. These steps are repeated for each processing frequency and model variant to gather information about each runtime configuration. This information guides the scheduler to accurately calculate the shares for each application and track their resource utilization.

In our implementation, the profiling overhead for the model-serving applications is in the range of tens of seconds. For applications that cannot be profiled with negligible overhead, we can leverage power models[7, 53] that estimate the energy required based on the application architecture and update their estimates as applications run and actual power consumption is measured. However, given the recent focus on energy consumption and carbon footprint of computing, there have been many efforts on benchmarking applications' power consumption[39]. We expect such effort to be more prevalent and profiling for most applications to be available, even for highly specialized use cases.

(a) Time **(b) Energy**

Figure 5: ***Energy and Time allocations of the two users under Time Fairness (TF), Energy Fairness (EF), and Energy Time Fairness (ETF) at*** $\phi = 0.7$.

6 EXPERIMENTAL EVALUATION

In this section, we empirically evaluate the practical performance of ETF by applying it to popular DNN model serving workloads. Our goal is to demonstrate how the theoretical properties of ETF (§4.4) translate into real-world benefits.

6.1 Experimental Setup.

Hardware. We conduct our experiments on Nvidia Jetson TX2 unless stated otherwise, which integrates an Nvidia Pascal GPU with 256 CUDA cores, a Dual-Core Nvidia Denver2, and a Quad-core ARM Cortex A57 CPU. It has 8GB memory with 59.7GB/s memory bandwidth and consumes a maximum power of 15W. Jetson TX2 supports GPU DVFS, with available GPU core frequency range from 115 MHz to 1300 MHz.

Workloads. Our experimental workloads primarily focus on the use case of DNN model serving. We use a diverse set of popular DNN models such as ResNet and MobileNet, each characterized by different sizes and energy profiles (figure 1). In our evaluation, users first load their respective models onto the GPU. Subsequently, they send inference requests to the model serving system driven by ETF. The results are transmitted back to the users after processing. We assume that workloads are backlogged and requests are always queued.

Baselines. We compare ETF with two widely used scheduling algorithms that use the notion of: 1) Time Fairness (TF), which ensures the equity of the process time across users, and 2) Energy Fairness (EF [13]), which is similar to TF but the resource that is to be shared fairly is the energy consumption.

Metrics. To quantitatively assess the fairness of allocations across applications, we use three metrics:

(1) **Time Fairness** $\mathcal{F}_{\mathcal{T}} = \frac{\min_{apps}\ time}{\max_{apps}\ time}$**:** It measures how evenly time is distributed among users, with a maximum value of 1 indicating perfect time fairness.

(2) **Energy Fairness** $\mathcal{F}_{\mathcal{E}} = \frac{\min_{apps}\ energy}{\max_{apps}\ energy}$**:** It measures the evenness of energy consumption among users, with a maximum value of 1 indicating perfect energy fairness.

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

Figure 6: ***System fairness for*** `ETF` ***(ϕ = 0.7) and baseline approaches at various levels of power disparity (decreases from left to right) between two applications. Application A_1 always operates at 1300MHz. The operating frequency of A_2 increases from 114MHz to 1300MHz.***

(3) **System Fairness $\mathcal{F}_S = \min(\mathcal{F}_\mathcal{T}, \mathcal{F}_\mathcal{E})$:** It captures the system-level fairness across both time and energy. The closer the value to 1, the fairer the allocation.

Here, both time and energy allocations are normalized by application weights to make them comparable across users.

6.2 Illustrating `ETF` in Action

As outlined in prior sections, EF and TF only allow pure energy and pure time fairness, respectively. `ETF` allows operating at a configurable point between the two extremes.

Figure 5 presents the time and energy allocations under `ETF` for a given value of ϕ and other baseline fair scheduling approaches. In our setup, application A_1 executes a `ResNet50` model running at a frequency of 1300 MHz, while application A_2 executes a `MobileNet-V2` model at a 726 MHz frequency. As shown, TF guarantees absolute time fairness and both applications get GPU access for an equal amount of time. However, this equal time allocation results in a substantial disparity in energy consumption, resulting in an energy fairness measure $\mathcal{F}_\mathcal{E}(TF) = 0.19$. Consequently, the overall system fairness for TF, $\mathcal{F}_\mathcal{S}(TF) = 0.19$.

On the other hand, EF operates in an inverse manner, achieving $\mathcal{F}_\mathcal{S}(EF) = 0.21$. `ETF` strikes a balance between energy and time fairness, achieving superior time fairness compared to EF and better energy fairness than TF. Consequently, it yields $\mathcal{F}_\mathcal{S}(ETF)$ of 0.42, which is 2× that of TF and EF.

Key takeaway: `ETF` *provides minimum time guarantees that are absent under EF while achieving better energy fairness than TF. As a result,* `ETF` *can achieve up to 2× improvements in system-level fairness over pure TF- or EF-scheduling.*

6.3 Mitigating the Effect of Power Disparity

The energy and time allocations are fair if all the applications operate at the power. However, as the disparity between power consumption increases, the energy and time allocations start to diverge under time- and energy-fair policies, respectively. `ETF` is able to mind this gap and its ability to bridge the gap is highest when the power disparity between the applications is maximum. To demonstrate this experimentally, we conduct an experiment with two applications using `ResNet50` model, where A_1 always operates at a fixed frequency of 1300MHz while we vary the operating frequency of A_2 varies between 114MHz and 1300MHz.

Figure 7: ***Energy and time allocations under*** `ETF` ***at varying values of ϕ. A_1 runs*** `ResNet50` ***at 1300 MHz and A_2 runs*** `MobileNet-V2` ***at 726 MHz on Jetson TX2.***

Figure 8: ***Energy and time allocations under*** `ETF` ***at varying values of ϕ. A_1 runs*** `GPT-2` ***and A_2 runs*** `MobileNet-V2` ***at maximum frequency on Nvidia A10G GPU on AWS.***

Figure 6 shows the system level fairness on y-axis for the three scheduling policies when power disparity decreases from 1-to-7.9 at 114MHz to 1-to-1 (no disparity) at 1300MHz on x-axis. At the highest power disparity, `ETF` outperforms both TF and EF by a factor of 2× and 1.6×, respectively. As the power disparity decreases, the fairness improvement decreases and all scheduling policies converge when there is no disparity.

Key takeaway: *The performance of* `ETF` *improves under adversity. The greater the disparity in power, the higher the improvement in system fairness, e.g., 2× at 1-to-7.9.*

6.4 Improving Energy Fairness under Time Guarantee

Our experiments so far have used a static value of the *time-fair* factor, ϕ, that configures the minimum time guarantee for applications. However, `ETF` exposes ϕ as a configurable knob that allows

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

Figure 9: ***Dynamic allocation of energy and time under*** `ETF` ***(ϕ = 0.7) for three applications that have different arrival and departure times. Applications run*** `ResNet50` ***at 1300, 1032, and 726 MHz GPU frequencies, respectively.***

selecting any values of minimum time and energy between the two extremes offered by EF and TF, respectively.

Figure 7 demonstrates the effect ϕ on energy and time allocations using the same experimental setup as Figure 5. The top figure shows the time allocations under `ETF` and the time allocated to the low power application (E_1) under EF. For ϕ values of less than 0.3, it is able to achieve perfect energy fairness for the application. For the remainder of ϕ values, `ETF` guarantees each application receives a minimum amount of time with respect to ϕ and maximizes energy fairness. This also empirically verifies theorem 3. The bottom figure shows the corresponding energy allocations that demonstrate that it achieves better energy fairness than *TF*. As expected, it reduces to pure time-fair scheduling at ϕ of 1.

Thus far, we have evaluated `ETF` using the Jetson TX2 platform. To affirm that our technique is adaptable and effective across various platforms, we repeat our experiments on the Nvidia A10G GPU. Figure 8 illustrates energy and time allocations when running a GPT-2 model concurrently with a `MobileNetV2` model, with all models operating at the same maximum GPU frequency. As shown, the result mirrors those obtained on the TX2, with the exception that the convergence point for energy fairness occurs at a higher ϕ value. This is due to a smaller power disparity in this scenario, which is approximately 2.2×, compared to the 4× on TX2.

Key takeaway: `ETF` *exposes ϕ that system operators configure to navigate the tradeoff between the energy and time allocations offered by the EF and TF, respectively. The effective range of ϕ depends on the power disparity across applications.*

6.5 Handling Dynamic Workloads

In the experiments so far, we had a static set of applications that ran for the whole duration of the experiment. However, in practice, applications will arrive and leave over time. `ETF` is robust to dynamic workload changes as it can quickly recompute the time allocated to each of the applications. To demonstrate that, we conduct an experiment with three applications A_1 to A_3, all running `ResNet50` model at GPU frequencies of 1300, 1032, and 726 MHz, respectively.

Figure 10: ***Throughput and latency for two applications running*** `ResNet50` ***model. A_1 always runs at 1300 MHz while A_2 runs at varying frequency given by x-axis.***

Figure 9 shows energy and time shares of the applications over time. Application A_1 runs throughout the experiment, but its share changes based on the presence of A_2 and A_3. When only A_1 and A_2 are present, A_1 is guaranteed $50\% \cdot \phi = 45\%$. Since the power disparity between A_1 and A_2 is not significant (1-to-1.5), `ETF` is able to mitigate the unfairness during the catch-up period. As A_3 enters the system, `ETF` has more work to do. `ETF` guarantees $33.3\% \cdot \phi \approx 23\%$ time share to each application. However, since the power disparity has increased to 1-to-2.5, `ETF` tries to mitigate the effect by assigning disproportionally high time share to low-power application A_3. However, in the remaining $0.3 \times T$ duration, it cannot mitigate the unfairness caused by the assignment of minimum time share to the most resource-hungry application. As a result, the energy shares are not exactly the same. It is also worth noting that `ETF` always uses 100% of GPU time, which confirms that `ETF` is Pareto efficient (theorem 1).

Key takeaway: `ETF` *provides minimum time guarantees while maximizing energy fairness when allocating resources to a dynamic set of applications in a pareto-efficient manner.*

6.6 Incentivizing Energy Efficiency

A key property of `ETF` is to incentivize energy efficiency. This happens because applications operating at lower power are rewarded with an increased time share, as demonstrated in the previous experiments. This incentivizes applications to operate at higher efficiency by operating at a different frequency under DVFS or by using a smaller model. The decrease in throughput from a lower operating frequency or a smaller model is compensated by the increase time share.

Figure 10 demonstrates the incentives for improving energy efficiency. We have two applications that operate at the same frequency (and power) in one scenario (left bars). In the second scenario, one of the applications (A_2) reduces its operating frequency (and power). As a result, application A_2 gets a higher time share than A_1 that yields 1.3× higher throughput while reducing the execution latency by 10%.

Key takeaway: `ETF` *rewards energy-efficient applications with a higher throughput and a lower execution latency, while providing minimum time guarantee to the inefficient application.*

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

6.7 Managing Application Weights

Our experiments so far have used equal weights across applications to simplify the results. In practice, applications can have different weights that indicate their priority to the system operator. ETF can handle any arbitrary weight allocations. To demonstrated that, we modify the experimental setup used in Figure 5 such that A_1 has twice the weight of A_2.

Figure 11 illustrates the energy and time allocations between A_1 and A_2. As depicted, when $\phi = 1$ (i.e. perfect time fair), A_1 receives twice the time share compared to A_2. When $\phi \leq 0.6$, ETF is able to achieve perfect energy fairness. In this case, A_1 receives twice the energy share as A_2. These results effectively demonstrate ETF's ability to respect the predefined application weights during the allocation process.

Key takeaway: *ETF can handle any arbitrary set of application weights while satisfying all of its properties.*

Figure 11: ***Energy and time allocations under ETF at varying values of ϕ. A_1 runs ResNet50 at 1300 MHz and A_2 runs MobileNet-V2 at 726 MHz with weights 2-to-1.***

7 RELATED WORK

Fair share scheduling The scarcity of resources in multi-tenant deployments made fair sharing key to ensuring the proper distribution of resources. In the context of a single resource, time-fair-sharing [12, 16, 26, 37, 43, 48] ensures key fairness objectives such as Pareto efficiency, strategy-proofness, and sharing incentives. In such work, the general assumption is that users utilize the allocated time in the same way, which makes energy just a utility and a factor of time. As we highlighted earlier, when applications run at different power levels, the *energy unfairness* of the presumably fair schedulers arises as highlighted in previous research [13, 19, 51]. Moreover, in many scenarios, such as battery-powered systems, energy is no longer a utility but a constrained resource that must be distributed fairly between applications. However, previous attempts to address the issue of energy unfairness were directed towards multi-resource environments [13, 19, 51]. As we showed earlier, these solutions could lead to starvation when applied to a single resource. Furthermore, as noted by ShapeShifter[38], the pursuit of time-fairness often comes at the expense of other performance metrics, such as SLO. In contrast, we present Energy Time Fairness — a configurable system designed to harmoniously reconcile both time and energy fairness.

Asymmetric-resource Fairness The presence of asymmetric resources, such as big-little CPU [17], is another example where the traditional notions of time fair sharing fail to distribute resources equally. The difference between resources capabilities and energy efficiency made progress and efficiency almost independent of the allocated time [27, 28, 30, 31, 34, 41, 47]. To address these discrepancies, authors devised new fairness mechanisms that users share the faster core equally [28, 30, 31] to ensure that no user hogs the fast resource. Authors highlighted the importance of fair distribution CPU cycles [47], energy [34, 41], and slowdowns [27]. On the other hand, we highlight that these techniques boil down to traditional time fair sharing or energy fair sharing when applied to a single resource.

Heterogeneous-resource Fairness The previous notions of fairness work only when resources are replaceable or exchangeable. However, in large-scale applications where a single resource type, such as CPUs, is not always readily available, traditional fair share schedulers can fail to provide the guarantees they promise [14, 15, 55]. To address these problems, the authors extended the concepts of fairness to multiple resources and show how fairness can be achieved in this context. DRF [15] introduced the concept of dominant resource, where the assumption is that application resource requirements are skewed. Although DRF targets space-sharing, several variations addressed the problem in the context of time-sharing. For example, DRFQ [14] extends the concept of time-fair sharing to multiple resources by altering the accounted allocation based on the most utilized resource. However, the key assumption of these solutions was that resources are independent, which makes these techniques inapplicable to the energy-time fairness problem. Our solution fixes these problems by implementing a hierarchical scheduler that addresses the dependency between time and energy resources.

8 CONCLUSION

Since traditional notions of fairness break down for newer heterogeneous hardware and applications, in this paper, we introduced a new notion of fairness called Energy-Time Fairness (ETF). Our approach aims to promote energy efficiency while ensuring that applications do not suffer from starvation by providing minimum time guarantees. We demonstrate that ETF satisfies many established fairness principles and successfully bridges the gap between energy and time fairness. We implemented ETF as a user-space GPU scheduler and showed that it supports both i) running time/latency-sensitive applications and ii) a configurable incentive for operating energy-efficiently. In the future, we plan to expand our system by creating a runtime library and adding support for other devices such as CPUs. As part of our future work, we plan to extend ETF to other types of accelerators and heterogeneous processors.

ACKNOWLEDGMENTS

We thank the SEC reviewers for their valuable comments, which improved the quality of this paper. This research is supported by NSF grants 2211302, 2211888, 2213636, 2105494, US Army contract W911NF-17-2-0196, Adobe, VMware, and Amazon Web Services.

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] 2023. Hadoop Fair Scheduler. https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/FairScheduler.html.

[2] 2023. Slurm Workload Manager Classic Fairshare Algorithm. https://slurm.schedmd.com/classic_fair_share.html.

[3] Noman Bashir, Tian Guo, Mohammad Hajiesmaili, David Irwin, Prashant Shenoy, Ramesh Sitaraman, Abel Souza, and Adam Wierman. 2021. Enabling Sustainable Clouds: The Case for Virtualizing the Energy System. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. Association for Computing Machinery, New York, NY, USA, 350–358. https://doi.org/10.1145/3472883.3487009

[4] Noman Bashir, David Irwin, Prashant Shenoy, and Jay Taneja. 2017. Enforcing Fair Grid Energy Access for Controllable Distributed Solar Capacity. In *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments* (Delft, Netherlands) *(BuildSys '17)*. Association for Computing Machinery, New York, NY, USA, Article 28, 10 pages. https://doi.org/10.1145/3137133.3137147

[5] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. 2021. A Comprehensive Survey on Hardware-Aware Neural Architecture Search. arXiv:2101.09336 [cs.LG]

[6] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems Over a Decade. *ACM Queue - Containers* 14, 1 (January-February 2016).

[7] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. 2017. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. *arXiv preprint arXiv:1710.05420* (2017).

[8] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-All: Train One Network and Specialize it for Efficient Deployment. arXiv:1908.09791 [cs.LG]

[9] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[11] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 613–627. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[12] A. Demers, S. Keshav, and S. Shenker. 1989. Analysis and Simulation of a Fair Queueing Algorithm. *SIGCOMM Comput. Commun. Rev.* 19, 4 (aug 1989), 1–12. https://doi.org/10.1145/75247.75248

[13] Yiannis Georgiou, David Glesser, Krzysztof Rzadca, and Denis Trystram. 2015. A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC. *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2015), 617–626.

[14] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. 2012. Multi-Resource Fair Queueing for Packet Processing. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Helsinki, Finland) *(SIGCOMM '12)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2342356.2342358

[15] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/nsdi11/dominant-resource-fairness-fair-allocation-multiple-resource-types

[16] Pawan Goyal, Harrick M. Vin, and Haichen Cheng. 1997. Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. *IEEE/ACM Trans. Netw.* 5, 5 (oct 1997), 690–704. https://doi.org/10.1109/90.649569

[17] Peter Greenhalgh. 2011. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White paper* 17 (2011).

[18] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 443–462. https://www.usenix.org/conference/osdi20/presentation/gujarati

[19] Daniel Hagimont, Christine Mayap Kamga, Laurent Broto, Alain Tchana, and Noel De Palma. 2013. DVFS Aware CPU Credit Enforcement in a Virtualized System. In *Middleware 2013*, David Eyers and Karsten Schwan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 123–142.

[20] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale Preemption for Concurrent GPU-accelerated DNN Inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 539–558. https://www.usenix.org/conference/osdi22/presentation/han

[21] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning Both Weights and Connections for Efficient Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 1135–1143.

[22] Walid A. Hanafy, Tergel Molom-Ochir, and Rohan Shenoy. 2021. Design Considerations for Energy-Efficient Inference on Edge Devices. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems* (Virtual Event, Italy) *(e-Energy '21)*. Association for Computing Machinery, New York, NY, USA, 302–308. https://doi.org/10.1145/3447555.3465326

[23] Soheil Hashemi, Nicholas Anthony, Hokchhay Tann, R. Iris Bahar, and Sherief Reda. 2017. Understanding the Impact of Precision Quantization on the Accuracy and Energy of Neural Networks. In *Proceedings of the Conference on Design, Automation & Test in Europe* (Lausanne, Switzerland) *(DATE '17)*. European Design and Automation Association, Leuven, BEL, 1478–1483.

[24] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *ArXiv* abs/1503.02531 (2015).

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. arXiv:1905.02244 [cs.CV]

[26] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. 1998. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *The Journal of the Operational Research Society* 49, 3 (1998), 237–252. http://www.jstor.org/stable/3010473

[27] Changdae Kim and Jaehyuk Huh. 2018. Exploring the Design Space of Fair Scheduling Supports for Asymmetric Multicore Systems. *IEEE Transactions on Computers* 67, 8 (2018), 1136–1152. https://doi.org/10.1109/TC.2018.2796077

[28] Youngjin Kwon, Changdae Kim, Seungryoul Maeng, and Jaehyuk Huh. 2011. Virtualizing performance asymmetric multi-core systems. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. 45–56.

[29] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. 2016. Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*. 477–484. https://doi.org/10.1109/BDCloud-SocialCom-SustainCom.2016.76

[30] Tong Li, Dan Baumberger, David A. Koufaty, and Scott Hahn. 2007. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. 1–11. https://doi.org/10.1145/1362622.1362694

[31] Tong Li, Paul Brett, Rob Knauerhase, David Koufaty, Dheeraj Reddy, and Scott Hahn. 2010. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. 1–12. https://doi.org/10.1109/HPCA.2010.5416660

[32] Qianlin Liang, Walid A. Hanafy, Ahmed Ali-Eldin, and Prashant Shenoy. 2023. Model-Driven Cluster Resource Management for AI Workloads in Edge Clouds. *ACM Trans. Auton. Adapt. Syst.* 18, 1, Article 2 (mar 2023), 26 pages. https://doi.org/10.1145/3582080

[33] Qianlin Liang, Walid A. Hanafy, Noman Bashir, Ahmed Ali-Eldin, David Irwin, and Prashant Shenoy. 2023. DěLen: Enabling Flexible and Adaptive Model-Serving for Multi-Tenant Edge AI. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation* (San Antonio, TX, USA) *(IoTDI '23)*. Association for Computing Machinery, New York, NY, USA, 209–221. https://doi.org/10.1145/3576842.3582375

[34] Ching-Chi Lin, Hsiang-Hsin Li, Jan-Jan Wu, and Pangfeng Liu. 2016. An Energy-Efficient Scheduler for Throughput Guaranteed Jobs on Asymmetric Multi-Core Platforms. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. 810–817. https://doi.org/10.1109/ICPADS.2016.0110

[35] Seyed Morteza Nabavinejad and Tian Guo. 2023. Opportunities of Renewable Energy Powered DNN Inference. arXiv:2306.12247 [cs.DC]

[36] Chandandeep Singh Pabla. 2009. Completely Fair Scheduler. *Linux J.* 2009, 184, Article 4 (aug 2009).

[37] A.K. Parekh and R.G. Gallager. 1993. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking* 1, 3 (1993), 344–357. https://doi.org/10.1109/90.234856

[38] Valentin Rakovic, Ke-Jou Hsu, Ketan Bhardwaj, Ada Gavrilovska, and Liljana Gavrilovska. 2022. ShapeShifter: Resolving the Hidden Latency Contention Problem in MEC. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. 237–251. https://doi.org/10.1109/SEC54971.2022.00026

[39] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.

Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 446–459. https://doi.org/10.1109/ISCA45697.2020.00045

[40] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 397–411. https://www.usenix.org/conference/atc21/presentation/romero

[41] Bagher Salami, Hamid Noori, and Mahmoud Naghibzadeh. 2021. Fairness-Aware Energy Efficient Scheduling on Heterogeneous Multi-Core Processors. *IEEE Transactions on Computers* 70, 1 (2021), 72–82. https://doi.org/10.1109/TC.2020.2984607

[42] Roy Schwartz, Jesse Dodge, Noah Smith, and Oren Etzioni. 2019. Green AI. *Commun. ACM* 63 (2019), 54 – 63.

[43] M. Shreedhar and G. Varghese. 1996. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking* 4, 3 (1996), 375–385. https://doi.org/10.1109/90.502236

[44] Abel Souza, Noman Bashir, Jorge Murillo, Walid Hanafy, Qianlin Liang, David Irwin, and Prashant Shenoy. 2023. Ecovisor: A Virtual Energy System for Carbon-Efficient Applications. In *ASPLOS*.

[45] M. Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ArXiv* abs/1905.11946 (2019).

[46] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2464–2469. https://doi.org/10.1109/ICPR.2016.7900006

[47] Kenzo Van Craeynest, Shoaib Akram, Wim Heirman, Aamer Jaleel, and Lieven Eeckhout. 2013. Fairness-aware scheduling on single-ISA heterogeneous multi-cores. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*. 177–187. https://doi.org/10.1109/PACT.2013.6618815

[48] Midhul Vuppalapati, Giannis Fikioris, Rachit Agarwal, Asaf Cidon, Anurag Khandelwal, and Éva Tardos. 2023. Karma: Resource Allocation for Dynamic Demands. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 645–662. https://www.usenix.org/conference/osdi23/presentation/vuppalapati

[49] Carl A Waldspurger and William E Weihl. 1994. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*. 1–es.

[50] Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu. 2020. ALERT: Accurate Learning for Energy and Timeliness. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 353–369. https://www.usenix.org/conference/atc20/presentation/wan

[51] Chengjian Wen, Jun He, Jiong Zhang, and Xiang Long. 2010. PCFS: Power Credit Based Fair Scheduler Under DVFS for Muliticore Virtualization Platform. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*. 163–170. https://doi.org/10.1109/GreenCom-CPSCom.2010.126

[52] Xen. 2018. Credit Scheduler. https://wiki.xenproject.org/wiki/Credit_Scheduler.

[53] Tien-Ju Yang, Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2017. A method to estimate the energy consumption of deep neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*. 1916–1920. https://doi.org/10.1109/ACSSC.2017.8335698

[54] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[55] Chaoqun You, Yangming Zhao, Gang Feng, Tony Q. S. Quek, and Lemin Li. 2023. Hierarchical Multiresource Fair Queueing for Packet Processing. *IEEE Transactions on Network and Service Management* 20, 1 (2023), 726–740. https://doi.org/10.1109/TNSM.2022.3197747

[56] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2018. Slimmable Neural Networks. arXiv:1812.08928 [cs.CV]

[57] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. 2020. Fast Hardware-Aware Neural Architecture Search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[58] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. arXiv:1611.01578 [cs.LG]

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on May 03,2024 at 14:00:58 UTC from IEEE Xplore. Restrictions apply.