Pareto Governors for Energy-Optimal Computing

RATHIJIT SEN and DAVID A. WOOD, University of Wisconsin-Madison

The original definition of energy-proportional computing does not characterize the energy efficiency of recent reconfigurable computers, resulting in non-intuitive "super-proportional" behavior. This paper introduces a new definition of ideal energy-proportional computing, new metrics to quantify computational energy waste, and new SLA-aware OS governors that seek Pareto optimality to achieve power-efficient performance.

CCS Concepts: •**General and reference** →**Metrics; Performance;** •**Hardware** →**Power and energy; Power estimation and optimization;**

Additional Key Words and Phrases: DVFS, prefetching, SLA, RAPL, performance-per-watt

ACM Reference format:
Rathijit Sen and David A. Wood. 2017. Pareto Governors for Energy-Optimal Computing. *ACM Transactions on Architecture and Code Optimization* 14, 1, Article 6 (March 2017), 25 pages.
DOI: 0000001.0000001

1 INTRODUCTION

> We see that peak energy efficiency occurs at peak utilization and drops quickly as utilization decreases.
>
> Luiz André Barroso and Urs Hölzle [3]

Energy efficiency is the work done per unit amount of energy used. Maximizing energy efficiency allows more work to be done for a given energy budget and also allows work to be done faster for a given power budget. This has economic and environmental benefits as it minimizes the energy needed to do a given computation.

Barroso and Hölzle [3] observed that real systems—at that time—attain peak efficiency at peak utilization, but quickly lose efficiency as utilization drops as they are unable to proportionately reduce power consumption. They posit that an "ideal" energy-proportional system should always use energy in proportion to the work done, by maintaining this peak efficiency even at reduced load.

Figures 1a and 1b illustrate this original model for an Intel Haswell server running SPECpower [36]. The points labeled with *Peak Performance Configuration* show the server's power-performance at different load levels with the highest processor frequency. The machine can serve maximum load (peak performance) with this configuration. The *EP* line represents Barroso and Hölzle's "ideal"

This work was supported in part by the National Science Foundation, under grant CCF-1218323, grant CNS-1302260, and grant CCF-1533885.
Author's addresses: R. Sen, Gray Systems Lab, Microsoft Corporation; D. A. Wood, Department of Computer Sciences, University of Wisconsin-Madison.

DOI: 0000001.0000001

energy-proportional profile where performance is linearly proportional to power. We consider this a *design ideal* for future systems, since current systems have unavoidable idle power consumption. The *Dynamic EP* line accounts for idle power [26], and represents an *operational ideal* for the current system. This server's Peak Performance Configuration achieves power-performance very close to Dynamic EP. Figure 1b shows that the corresponding energy efficiency (η), normalized to that at peak performance, reduces quickly from 100% as performance drops. In contrast, an EP system is always 100% efficient.

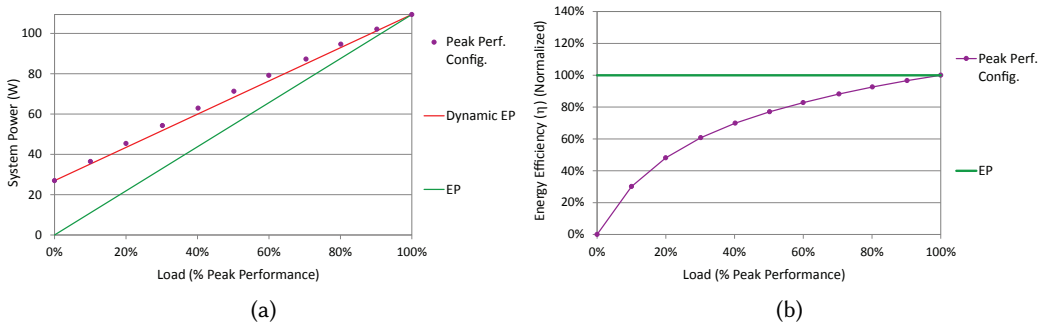


Fig. 1. (a) Power vs Load and (b) Efficiency vs Load for conventional systems.

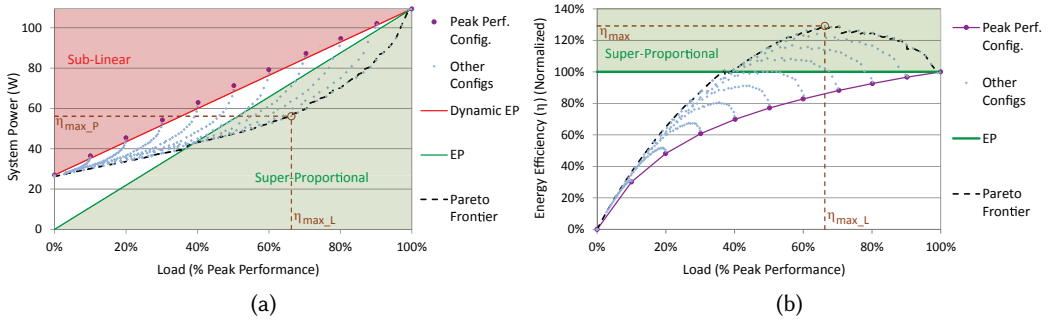


Fig. 2. (a) Power vs Load and (b) Efficiency vs Load for super-proportional systems.

Barroso and Hölzle's observation has been instrumental in helping drive recent system designs to have lower idle power and a wide dynamic power range. However, their model describes systems with *fixed resources*, while these modern, more-efficient processors have *reconfigurable resources*—e.g., core frequencies, voltages, number of active cores, threads per core, etc. that can be varied at runtime. Operating such a server with fixed resources can be inefficient when it faces variable loads. Load variation can occur due to fluctuating demands, or service consolidation and load balancing among other servers [7, 9, 28].

Servers achieve maximum performance when configured for peak performance (Peak Performance Configuration), but other configurations can trade performance for greater energy efficiency. Figures 2a and 2b show that changing just the socket frequency (and consequently voltage) results in energy efficiency that exceeds the "ideal" EP profile. Specifically, by varying the frequency from 3.9 to 0.8 GHz, the server can achieve super-proportional efficiency over almost 60% of the performance

range (points in the shaded Super-Proportional region—where performance is super-proportional to power). Figure 2b shows that the maximum efficiency, η_{max}, is 29% higher relative to the EP energy efficiency for this server. η_{max} occurs at load η_{max_L} (which is approximately two-thirds of maximum load) and power consumption η_{max_P}.

Reconfigurable systems create opportunities for increased efficiency even outside the super-proportional region. For example, Figure 2b shows that the Peak Performance Configuration attains a relative efficiency of 61% at 30% load, while a different configuration achieves a relative efficiency of 88% at the same load. In other words, the usual server configuration uses 44% more energy than necessary to satisfy the same load, despite being nearly on the Dynamic EP line.

Thus, neither EP nor Dynamic EP (that is, the conventional ideal models) describes the full potential of modern computing systems. While non-linearity with reconfiguration is well-known, e.g., with frequency (and voltage) control, the existing ideal models do not consider its impact on peak efficiency. We propose new ideal models to address this limitation.

Conventional energy proportionality metrics are defined using conventional ideals and have limitations for super-proportional systems. A popular metric, *EP_metric*[1] [32, 45], compares the area between the power-performance curves of a real server relative to that of the "ideal" energy-proportional server to quantify how close the real server is to the "ideal" server. Depending on the shapes of their power-performance curves, a super-proportional server can have a *EP_metric* value close to that of a sub-proportional server. Thus, *EP_metric* does not identify all servers that can benefit from scheduling policies that exploit super-proportionality (e.g., policies other than race-to-halt).

Recent works [30, 44] have proposed schedulers that can exploit super-proportionality, but continue to use the conventional ideals for energy proportionality. Differing goals and ideals can create confusion in scheduling applicability. For example, although Wong [44] observes that the peak efficiency is higher than the "ideal" (conventional energy proportional) efficiency, and occurs at intermediate utilizations, they propose the Peak Efficiency Aware Scheduler (PEAS) "for highly energy proportional servers" (identified using *EP_metric*). Our redefined notion of energy proportionality removes this semantic ambiguity. In our framework the ideal model achieves peak efficiency.

EP_metric does not quantify excess energy used by the real server compared to the least possible at a given load level. Another metric, Proportionality Gap (PG) [45], is parametrized by the utilization level, $x\%$, and is defined as:

$$PG_{x\%} = \frac{Power_{actual@x\%} - Power_{ideal@x\%}}{Power_{peak}}$$

For a conventional non-reconfigurable server, $PG_{x\%} \geq 0$. However, for a reconfigurable super-proportional server, PG turns out to be negative for utilizations in the super-proportional region if the conventional ideal model is used. We find this mix of positive and negative values non-intuitive for a metric to quantify room for improvement in energy proportionality of an actual server with respect to an ideal model.

Another limitation of PG is that it does not quantify the load allocation and server configuration aspects of suboptimality in attained energy efficiency. Separation of these aspects is useful since energy efficiency for a server in a cluster can be affected both by inter-server load distribution as well as intra-server configuration selection decisions. Our new metrics quantify these effects for each server. Together they influence how much computational energy a server is using relative to its optimum.

[1]The metric is called EP in the cited papers, but we use *EP_metric* in this discussion to differentiate it from the EP model.

This paper describes an analytical framework that integrates the concepts of energy proportionality, energy optimality, Pareto efficiency, peak efficiency, ideals for system designers and operators, lower bounds on energy consumption, and develops new mechanisms that meet user-specified Service-Level Agreements (SLAs) while minimizing computational energy.

The main contributions of this work are:

(1) **New Ideals**: We propose new ideals for both system designers and system operators. Energy Optimal Proportional (EOP) is the new design ideal that subsumes conventional "ideal" energy proportionality. Dynamic Energy Optimal (Dynamic EO), that is the power-performance Pareto frontier, is the new operational ideal.
(2) **New Metrics**: We propose a new metric called Computational Power Usage Effectiveness (CPUE) to quantify excess computational energy used with respect to that by EOP. We also propose new metrics, Load Usage Effectiveness (LUE) and Resource Usage Effectiveness (RUE), that can help system operators to focus on load management and configuration management to make the system operate efficiently. We develop the "Iron Law of Energy" that quantifies the impact of poor load management (affecting LUE) and poor configuration management (affecting RUE) on CPUE.
CPUE mirrors the well-known datacenter-level PUE metric [1] for a server's computational energy consumption. Just as datacenter operators aim to reduce PUE, server operators/schedulers should try to reduce CPUE either by reducing LUE or RUE or both. Scheduling policies can reach target CPUE levels by governing for one or both of these aspects after taking into account additional constraints such as data movement costs for stateful services.
(3) **New Governors**: We develop new OS governors that seeks Pareto-optimality for socket frequency (and associated voltage) scaling and hardware cache prefetching. OS governors currently do not control hardware prefetching.

2 TERMINOLOGY, INFRASTRUCTURE, AND WORKLOADS

Similar to Barroso and Hölzle [3, 4], we define energy efficiency as $\frac{\text{Work}}{\text{Energy}}$, or equivalently, $\frac{\text{Performance}}{\text{Power}}$. The performance of a system is measured as the rate of doing work, e.g., the load served, transactions completed, or instructions executed per unit time. We refer to 100% load as the maximum load achieved for the Peak Performance Configuration (all cores at the highest frequency and prefetching enabled). All loads are normalized with respect to that peak load. The normalized load is the system utilization [3].

The system that we use is a single-socket quad-core Haswell Xeon E3-1275 v3 server with 32 GB memory (DDR3-1600), henceforth referred to as HS. HS runs RHEL 2.6.32. It has a frequency range of 0.8–3.9 GHz with 3.5^{+}–3.9 GHz being the turbo boost region. The turbo boost plan is 2/3/4/4 meaning that the maximum frequency can be 3.5 + 0.1*2 = 3.7 GHz with all four cores active, 3.5 + 0.1*3 = 3.8 GHz with three cores active, and 3.5 + 0.1*4 = 3.9 GHz with two or one cores active. We run the system with all four cores, hyperthreading (thus, 8 threads per socket), and cache prefetching enabled by default. All cores run at the same frequency (HS does not support per-core DVFS). The socket frequency can changed in steps of 100 MHz by writing to Model Specific Registers (MSRs). Any value for the turbo region limits the maximum frequency. HS has a socket TDP of 84W, socket idle power of 0.27W, and DRAM idle power of 4.3W.

The OS acpi-cpufreq interface allows controlling frequency in 15 steps from 0.8–3.5 GHz (0.8–2.0 GHz and 2.1 GHz–3.5 GHz in steps of 200 MHz) and enabling/disabling the turbo boost region (3.5^{+} GHz). We use this granularity while comparing with the existing Linux governors (Section 9 onwards). We use the finer granularity of 100 MHz elsewhere and also in Section 11.3.

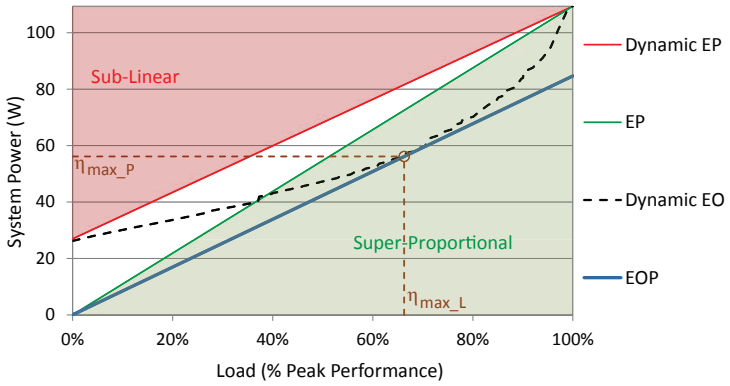


Fig. 3. EOP and Dynamic EO models.

We measure socket power and DRAM power using an additional software thread that reads available RAPL (Runtime Average Power Limit) counters [19, 21] at 1 second intervals. This runs as a thread separate from application threads and any governor threads. The governors that we develop also read RAPL counters for power calculations. We measure wall power with a Watts Up? (.net) meter [12] at 1 second intervals. This also runs as a separate additional thread for experiments where we use wall power.

We run the SPECpower benchmark [36]. This Java workload simulates warehouse transaction processing, with (by default) as many warehouses as logical processors on the system under test, that is, the server. Transaction requests to each warehouse arrive in batches of 1000 transactions each. The batches have (negative) exponentially distributed interarrival times. The server load is measured in total transactions per second. The workload first calibrates the maximum, or 100%, load. Next, it does measurement intervals by varying the load offered to the system under test from 100% (max. utilization) to 0% (no utilization) in decrements of 10%. In these intervals, the load served must be within 2% (up to 2.5% shortfall for the 100% and 90% intervals is allowed) of the offered load. SPECpower uses its own software daemon to periodically measure and report system power in the measurement intervals.

We also run graph500 [14], hpcg [11, 33], and 14 workloads from SPECOMP2012 [37]. graph500 and SPECpower are run fully whereas the other workloads are run for the first 1200 seconds (a few runs of kdtree complete within this time at high frequencies).

3 REDEFINING EP AND DYNAMIC EP

The EP model assumes that maximum energy efficiency occurs at maximum (100%) load and argues that an ideal system should achieve that efficiency for all loads. Yet Figure 2b shows that a reconfigurable server actually attains maximum efficiency (η_{max}) at a lower load ($\eta_{max\_L} < 100\%$). We argue that a better ideal model is one that achieves this optimal efficiency η_{max} for all loads.

Similar to the EP model, the ideal system should have maximum efficiency (η_{max}) at every load. This implies that for a given computation, it will use minimum energy (E_{min}) to do it irrespective of the computing rate (performance or load). Figure 3 shows its geometric interpretation as a straight line passing through the points (0, 0) and (η_{max_L}, η_{max_P}). This ideal system, that is energy optimal at every load, uses power linearly proportional to load (l/η_{max} power at load l). Energy optimality at every load implies energy proportionality, but the converse is not true, e.g., EP is proportional but not optimal at all loads.

We call this new model *EOP* (Energy Optimal Proportional) since it is both optimal and proportional. EOP is a *design ideal* that gives system designers a way to measure how far the energy efficiency of a target design differs from the best possible design, hopefully leading to more energy-efficient systems. EOP subsumes the EP model—it improves upon EP for super-proportional systems and is identical to it for all others.

Of course real systems are unlikely to achieve this design ideal, e.g., due to unavoidable idle power, so system software needs an operational model that characterizes the maximum efficiency that can be realized by the current system at different loads. We address this using the well-known power-performance *Pareto frontier* [2, 5, 31], shown as a dashed line in Figures 2a–4. The Pareto frontier represents configurations in the current system that use the lowest power, and hence are the most efficient, among all configurations that can serve a given load. These configurations are Pareto optimal in the sense that, among these configurations, one cannot reduce power without also reducing load or increase load without also increasing power.

We call this model *Dynamic EO*. Like Dynamic EP, it is an operational ideal that seeks to characterize the best energy efficiency that can be achieved for a given system. But it differs from Dynamic EP in two aspects—it characterizes optimality that can already be realized by some among the multitude of configurations in the current system and it does not assume linearity of the power-performance profile.

Figure 3 illustrates the different models. These are the

- *design ideals*: conventional (EP), new (EOP), and
- *operational ideals*: conventional (Dynamic EP), new (Dynamic EO).

The EOP line tangentially meets the Dynamic EO line only at points having the maximum efficiency (η_{max}). The following energy efficiency relations hold for any system:

$$\text{Dynamic EP} \leq \text{EP} \leq \text{EOP}$$
$$\text{Dynamic EO} \leq \text{EOP}$$

where $\leq$ means less than or equal to for values of efficiency. Systems, like our server, that can operate in the non-Sub-Linear region for any portion of their performance range have Dynamic EP $\leq$ Dynamic EO for all such loads.

4 POWER-PERFORMANCE PARETO FRONTIER (DYNAMIC EO)

Every configuration of the system can be characterized by its performance and power consumption. We call each such (Configuration, Performance, Power) tuple a system state. The Pareto frontier is determined by only those states that use the lowest power among all states having at least that performance. It is a subset of the set of system states. Our governors seek to constrain system operations to the Pareto frontier.

Let Π denote the set of system states with Π_i representing the i^{th} state having performance $\Pi_i.Perf$ and power consumption $\Pi_i.Power$. Let the highest performing state be Π_0. We apply the well-known concepts of Pareto dominance and Pareto optimality. State Π_i Pareto-dominates state Π_j if $(\Pi_i.Perf \geq \Pi_j.Perf) \wedge (\Pi_i.Power \leq \Pi_j.Power)$.

Property 1: *The Pareto frontier is the set of non-dominated states.*

In Figures 2a and 2b, the Pareto frontier is the set of states represented by the dashed line. The states that lie on the EP line in the Super-Proportional region are dominated by the states on the Pareto frontier.

Implication: Constraining system operation to the Pareto frontier is important since dominated states are less efficient than dominating states (also see Figure 2b). The state with the maximum efficiency (η_{max}) lies on the Pareto frontier.

Property 2: *States on the Pareto frontier have the same total order in both power and performance.*

Let Π_i, Π_j be states on the Pareto frontier. Then $(\Pi_i.Perf > \Pi_j.Perf) \iff (\Pi_i.Power > \Pi_j.Power)$. We number the states in decreasing order of performance. The ordering relation for states on the frontier is thus: $i < j \iff (\Pi_i.Perf > \Pi_j.Perf) \wedge (\Pi_i.Power > \Pi_j.Power)$.

Implication: While the state space is two-dimensional, the Pareto frontier is more constrained allowing system operators to qualitatively reason about the other dimension from looking at one dimension alone. For example, increasing the power budget *will* improve performance at the Pareto frontier if the power is used. This is not true for the whole state space where states with less performance can use more power. This positive correlation between the two dimensions exists at the Pareto frontier.

Property 3: *System states that optimize power-performance metrics are located at the Pareto frontier.*

Consider a state Π_i that is not on the frontier. So there exists at least one other state Π_j such that $\Pi_i.Perf \geq \Pi_j.Perf$ and $\Pi_i.Power \leq \Pi_j.Power$ with at least one of the inequalities being strict. This implies that the highest performing state with/without a (maximum) power cap and the lowest power state with/without a (minimum) performance bound lie on the Pareto frontier.

In this work we assume that $performance \propto delay^{-1}$. Since energy is power multiplied by time (delay), it implies that the lowest energy point with/without a delay cap must lie on the Pareto frontier. Since the state corresponding to the highest performance-per-watt is the same as the state with the lowest energy, that state will also be on the Pareto frontier. Moreover, according to the above condition, states corresponding to the minimum energy-delay (ED) product or ED^2 product or, in fact, any $ED^n, n \geq 0$ must also lie on the Pareto frontier.

Since Pareto-optimal states are more efficient than other states, the highest performing state with/without a maximum power cap, the lowest power state with/without a minimum performance bound, the highest performance-per-watt state, the lowest energy state, the lowest energy-delay state, etc. will lie on the Pareto frontier.

Implication: Optimizing system operations for commonly used power-performance or energy efficiency metric necessitates operating it at the Pareto frontier.

Property 4: *The points of contact between the frontier, $Power = f(Perf)$, and the tangent curve $Power = c_n(Perf)^{n+1}, n+1 \geq 0$ and some constant c_n, represent configurations that optimize (minimize) metric ED^n. ($n = 0$ means energy E.)*

Let Π_i be a state that optimizes (minimizes) metric ED^n. By Property 3, Π_i must be on the frontier. Since $E = Power(Perf)^{-1}$ and $ED^n = Power(Perf)^{-n-1}$, Π_i will be on the curve for the power function $Power = c_n(Perf)^{n+1}$ if we choose $c_n = \Pi_i.Power(\Pi_i.Perf)^{-n-1}$. c_n is thus the optimum value for ED^n. Moreover, every point on this power function curve will have the same value for ED^n, which is c_n. No part of the frontier can be below this curve, as then states on this part of the frontier will have lower power for the same performance compared to points on the power function curve directly above them and thus have a smaller value for ED^n than c_n which is a contradiction.

All points on the curve above the linear tangent are suboptimal with respect to E, all points above the quadratic tangent are suboptimal with respect to ED, all points above the cubic tangent are suboptimal with respect to ED^2, and so on.

Implication: This forms the basis for the geometric interpretation of the EOP line described in Section 3. Every point on the linear tangent has the same slope, which is equal to $\frac{\text{Power}}{\text{Performance}}$, that is, performance-per-watt^{-1} value of the most energy-efficient point.

Property 5: *The Pareto frontier is not necessarily convex (or concave).*

Let Π_i, Π_j, Π_k be states on the frontier with $i < j < k$. The ordering relations only imply $\Pi_i.Perf > \Pi_j.Perf > \Pi_k.Perf$ and $\Pi_i.Power > \Pi_j.Power > \Pi_k.Power$, not $\Pi_j.Power \leq \Pi_k.Power + \left(\frac{\Pi_j.Perf - \Pi_k.Perf}{\Pi_i.Perf - \Pi_k.Perf}\right)(\Pi_i.Power - \Pi_k.Power)$.

Implication: Convex optimization approaches cannot be directly applied while composing multiple Pareto frontiers. Moreover, hill-climbing based search techniques at the frontier can get stuck in local optima instead of reaching global optima. However convex (polynomial) approximations to the Pareto frontier may work well.

5 COMPUTATIONAL PUE

Datacenters can satisfy a given load by distributing it to machines in different ways. Each machine can also be configured in a large number of ways. These modes for servicing the load differ in the amount of energy consumed, since some modes are more inefficient than others. A hypothetical ideal system, that is, one that meets the design ideal EOP, achieves maximal energy efficiency (η_{max}) and thus minimizes the energy (E_{min}) needed for a given computation regardless of load. We would like a metric to quantify the excess energy used by a real system, compared to this ideal system.

Our new metric, *Computational Power Usage Effectiveness* (or, CPUE), measures how much energy a server uses with configuration c at load l compared to the energy used by EOP. We define

$$\text{CPUE}(c,l) = \frac{\text{Actual server energy with } c \text{ at } l}{\text{EOP energy at } l}, \qquad l > 0 \tag{1}$$

$$= \frac{E(c,l)}{E_{min}}, \qquad l > 0 \tag{2}$$

$$\text{Thus, } E(c,l) = \text{CPUE}(c,l) \times E_{min}, \qquad l > 0 \tag{3}$$

$\text{CPUE}(c,l)$ is inspired by the well-known PUE metric [1] that tracks energy waste for datacenters by taking the ratio of facility energy consumption to energy consumption by IT equipment. PUE > 1 quantifies excess relative energy used by the datacenter due to the non-IT infrastructure. Similarly, $\text{CPUE}(c,l) > 1$ quantifies excess relative computational energy used whenever efficiency drops below η_{max}.

We defined $\text{CPUE}(c,l)$ as $E(c,l)/E_{min}$. We will now decompose $\text{CPUE}(c,l)$ to focus on two major factors that cause inefficiencies: i) running the system at a non-optimal load and ii) for a given load, running the system with a non-optimal configuration. For a given amount of work, energy consumed is inversely proportional to efficiency. Thus,

$$\text{CPUE}(c,l) = \frac{\eta_{max}}{\eta(c,l)}, \qquad l > 0 \tag{4}$$

$$= \left(\frac{\eta_{max}}{\eta_{\text{Pareto}}(l)}\right) \times \left(\frac{\eta_{\text{Pareto}}(l)}{\eta(c,l)}\right), \qquad l > 0 \tag{5}$$

$$= \text{LUE}(l) \times \text{RUE}(c,l), \qquad l > 0 \tag{6}$$

$$\text{Thus, } E(c,l) = \text{LUE}(l) \times \text{RUE}(c,l) \times E_{min}, \qquad l > 0 \tag{7}$$

where $\text{LUE}(l)$ denotes *Load Usage Effectiveness* at load l and $\text{RUE}(c,l)$ denotes *Resource Usage Effectiveness* of configuration c and load l.

$\text{LUE}(l)$ is the efficiency of EOP (η_{max}) relative to that of of Dynamic EO at load l. $\text{LUE}(l) \geq 1$ with $\text{LUE}(l) = 1 \iff l$ can be served at maximum efficiency (η_{max}). Since energy consumed is inversely proportional to efficiency, $\text{LUE}(l) > 1$ quantifies excess energy consumed, relative to E_{min}, due to non-optimal loads assuming that the Pareto-optimal configuration is used to serve load l.

$\mathrm{RUE}(c,l)$ is the efficiency of Dynamic EO relative to that of configuration c, both at load l. $\mathrm{RUE}(c,l) \geq 1$ with $\mathrm{RUE}(c,l) = 1 \iff c$ is a Pareto-optimal configuration. $\mathrm{RUE}(c,l) > 1$ quantifies excess energy used, relative to Dynamic EO at load l, due to using non-optimal (Pareto-dominated) configuration c for serving load l.

Our proposed RUE and LUE metrics can help system operators isolate the sources of energy inefficiency and guide new policies to reduce it. LUE is important for load management of Pareto-optimal configurations. RUE is important for configuration management for Pareto-dominated configurations. While LUE is applicable to all systems, both old and new, it only partially quantifies energy waste in reconfigurable systems that can be configured in a plurality of ways. RUE completes the quantification. Both $\mathrm{LUE}(l)$ and $\mathrm{RUE}(c,l)$ can be expressed in terms of $\mathrm{CPUE}(c,l)$. Since $\mathrm{RUE}_{\mathrm{Pareto}}(l) = 1$ for every l, $\mathrm{LUE}(l) = \mathrm{CPUE}_{\mathrm{Pareto}}(l)$ and $\mathrm{RUE}(c,l) = \mathrm{CPUE}(c,l)/\mathrm{CPUE}_{\mathrm{Pareto}}(l)$.

Inspired by the "Iron Law of Performance", we call Equation 7 the "Iron Law of Energy" to help with holistic management of server energy consumption. System designers will focus on minimizing $\mathrm{E}_{\mathrm{min}}$ whereas system operators will focus on minimizing LUE and RUE.

6 LOAD AND CONFIGURATION MANAGEMENT

Most data centers are provisioned to meet peak load, but normally operate at much lower load levels. The LUE metric can help operators quantify the potential benefit of deploying load management policies [7, 9, 28, 44], e.g., concentrating load on some servers and shutting down others. Of course, any such policy must also ensure that service-level agreements are still satisfied [30].

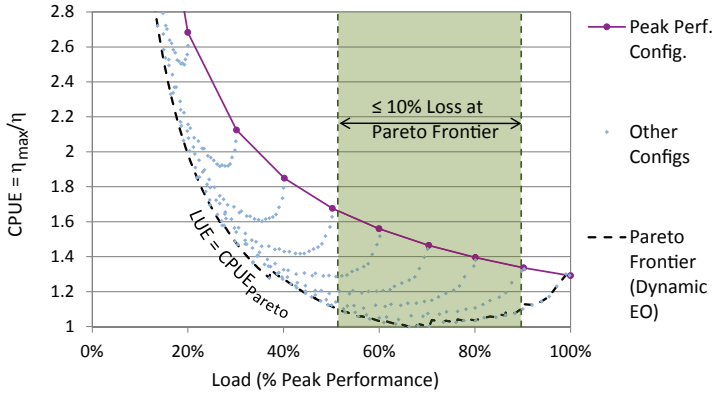


Fig. 4. $\mathrm{CPUE}(c,l)$ and $\mathrm{LUE}(l)$. These $\to \infty$ as load $l \to 0$ due to non-zero idle power. For any configuration c and load l, $\mathrm{CPUE}(c,l) \geq \mathrm{CPUE}_{\mathrm{Pareto}}(l) = \mathrm{LUE}(l) \geq 1$.

Figure 4 shows that CPUE for the Peak Performance Configuration is always > 1 (wastes energy) and increases as load decreases. The best CPUE for this configuration is 1.29, occurs at peak load, and implies 29% excess energy used relative to $\mathrm{E}_{\mathrm{min}}$. LUE (that is, CPUE for Dynamic EO), on the other hand, first decreases to 1, then increases, revealing a sweet spot of $\leq 10\%$ excess energy used at around 51%–90% of peak performance.

Barroso and Hölzle [3] observed that servers typically operate at 10%–50% load. The LUE curve for SPECpower (Figure 4), shows excess energy used due to suboptimal load of approximately 10% at the higher end of this range, to over 250% (not shown) at the lower end. The steep slope of the LUE curve at low loads makes even modest load management very attractive. For example,

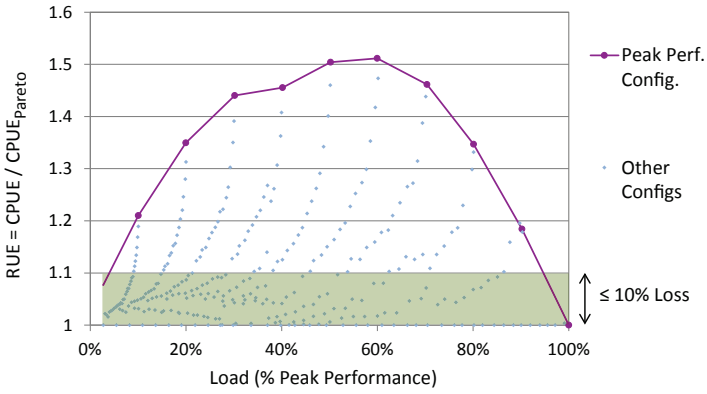


Fig. 5. RUE(c,l).

increasing load from 10% to 20% of peak reduces LUE from 3.55 (255% excess) to 1.99 (99% excess) and a further increase to 25% peak load reduces LUE to 1.68 (68% excess).

Even in a data center with perfect load balancing, reconfigurable servers may be misconfigured, wasting significant energy even at optimal load. Figure 5 shows RUE for SPECpower for all system configurations and loads. Operating with the Peak Performance Configuration is significantly wasteful even at low loads, e.g., 21% excess energy used at 10% load compared to operating at Dynamic EO. The excess increases to 51% before decreasing to zero at peak load. Not all Pareto-dominated configurations are as wasteful—the shaded band identifies configurations that have an RUE of ≤ 1.1 and hence limit the extra energy used to 10%.

7 PARETO GOVERNORS

We now develop new operating system governors (resource managers) that seek to operate the system at or close to Dynamic EO (power-performance Pareto frontier) so that Service-Level Agreements (SLAs) are satisfied. We call such governors *Pareto governors*. We consider the following SLAs in this work.

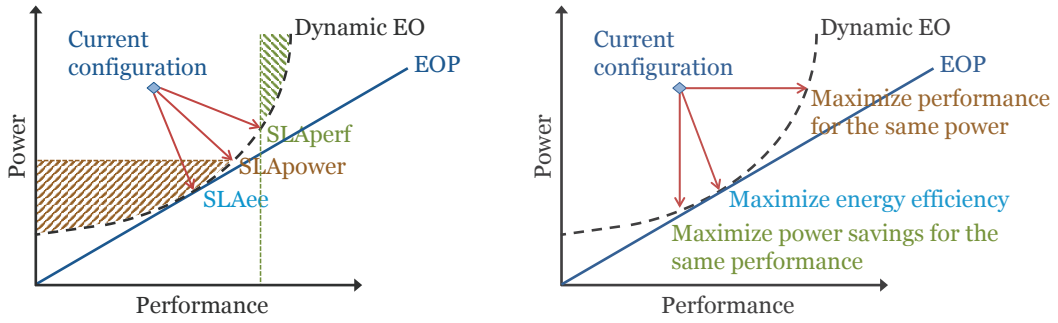
- **SLAee**: Maximize energy efficiency.
- **SLApower**: Maximize performance given a power cap/budget.
- **SLAperf**: Maximize power savings given a performance target.

Figure 6 shows what transitions the Pareto governors must make to the current operating point to meet the SLAs. The shaded regions in Figure 6a depict subsets of the system state space that satisfy a given power cap (for SLApower) or performance bound (for SLAperf). Figure 6b shows a special case for SLApower and SLAperf. The special case of maximizing power savings for the same performance affects only RUE but not LUE.

Our new governors, that meet these SLAs, have the following two-level design:

(1) **Pareto Predictor**: This predicts the power-performance Pareto frontier for the system and currently observed execution profile.
(2) **Objective Selector**: This level selects the desired operating state from the Pareto frontier according to the SLA to be achieved.

The objective selector remains unchanged if the available knobs change and the Pareto predictor remains unchanged if new SLAs are targeted. This separation, due to Property 3 (Section 4) of the Pareto frontier, simplifies design and enhances portability.



(a) General case (b) Special case for SLApower and SLAperf.

Fig. 6. State transitions to Dynamic EO for meeting SLAs.

Our governors use the BIPS (Billion Instructions Per Second) throughput metric to determine current application or to set performance targets. We assume the existence of user-supplied software routines that convert between BIPS and high-level performance metrics. Similar to existing governors in Linux, our new governors do not keep track of higher-level application constructs such transactions, queries, etc. The workloads that we consider for our experiments do not have any latency constraints. Other workloads that have latency constraints on high-level constructs should estimate their BIPS requirement and communicate that to the governors.

8 EXISTING GOVERNORS IN LINUX

The Linux acpi-cpufreq module includes the following governors [6] that control the operating frequency. The goal is to manage power-performance by either setting the frequency statically, or by varying it in response to processor utilization. With root privileges, the user can (dynamically) change the governor.

- **PowerSave** (**S**): Sets all cores to the lowest frequency. The idea is to use the least amount of power to do the work, but performance may be less than what could be achieved on this machine.
- **OnDemand** (**O**): Periodically samples (default: 10 ms interval) cores to adjust frequencies based on core utilization. The idea is to reduce power by lowering frequency when the CPU is not fully utilized and increase frequency as utilization increases so that the performance impact is minimal. The **Conservative** (**C**) governor is a variant of the OnDemand governor with more conservative utilization thresholds for changing frequencies.
- **UserSpace** (**U**): The idea is to give the user (having root privileges) control of the frequency settings. On HS, only the socket frequency (all cores together) can be set.
- **Performance** (**P**): Sets all cores to the highest frequency. The idea is to get the maximum performance. This governor also uses the maximum power.

To further distinguish between modes, we constrain **U** mode to exclude **S** or **P** mode frequencies, i.e., it operates in the range of 1.0–3.5 GHz. While these governors attempt to control knobs (e.g., processor frequency) in the system, none of them seek to meet SLAs that deal with energy consumption, power limits, or performance targets.

9 SLAEE: MAXIMIZE ENERGY EFFICIENCY

Temporarily, we consider system power as the sum of socket power and DRAM power, both of which are estimated using RAPL counters.

HS exhibits significant opportunities in improving BIPS/Watt (equivalently, Instructions/nanoJoule) by changing frequency settings alone. BIPS changes between 1.18x (swim) to 4.86x (bwaves) in going from **S** to **P** modes whereas power changes between 2.52x (swim) to 5.67x (botsalgn), leading to a BIPS/Watt range of 1.29x (imagick) to 2.14x (swim) between best and worst values for that workload over all frequencies. For all these workloads, the minimum BIPS/Watt happens for **P** mode. applu and graph500 show a BIPS/Watt range of 1.84x and 1.67x respectively (also see Figure 7). We study SPECpower for SLAperf, but not for SLAee, as it must meet performance constraints.

To exploit the improvement potential, we propose a simple reactive, **R(*t*)**, mode of operation. Our approach is to sample power and performance at a few different frequencies, then use that information to interpolate the frontier. The Pareto frontiers are usually non-linear. Thus, at least three samples are needed to target super-proportionality. In contrast, aiming for proportionality would require only two points, but the non-linearity in system behavior between the points could not be predicted or controlled.

We implement two power-performance predictors (in software)—one for the socket subsystem and the other for the memory subsystem. The socket predictor sets the frequency to 0.8 GHz (lowest frequency), 2.1 GHz (midpoint frequency) and 3.5 GHz (nominal frequency) in three consecutive intervals of t ms each and observes the power, performance, memory read and write bandwidths for each setting. It then interpolates the effects for the other frequencies.

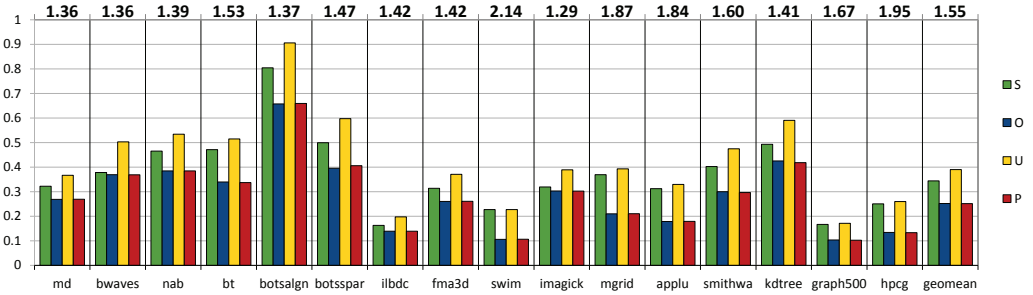
A software coordination module, running on one of the cores, reads the socket predictions and DRAM predictions every $51t$ ms (immediately after the $3t$ socket sampling), composes the predictions, estimates the frontier and selects the best frequency. The length of the interval that the system runs in this state is $48t$. It is during this time that the DRAM predictor is periodically invoked (every $12t$ ms) to adjust a computed linear regression between DRAM power and read and write bandwidths (two variables) based on current readings. The regression is reset every 17 observations ($204t$ ms) to react faster to phase changes. We choose this value since 17 is not divisible by 4 ($48t/12t = 4$), so the regression will not be reset at the time when the readings are needed for the power estimations with the interpolated values.

Since the optimal frequency will be in of the high/mid/low ranges, the sampling overhead is approximately $\frac{2t}{51t} \approx 4\%$. The workload continues to execute, although suboptimally, in those two intervals, so the overhead is usually $< 4\%$. For the intervals mentioned above, we do not account for additional governor overheads due to system calls, interpolation, estimations, etc. So, the actual intervals are slightly longer. The governor in Section 11.3 accounts for these overheads.

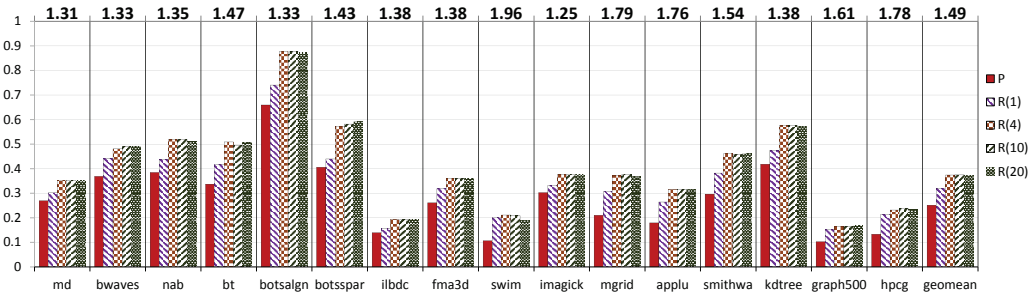
There are three issues in implementing the interpolant for the socket predictor—getting successive samples having non-decreasing performance and power with increasing frequency, getting samples with acceptable measurement noise/jitter, and dealing with non-convexity of the frontier. The first issue arises when the workload shows local phase behavior. The second issue arises with rapid sampling that makes the jitter in the energy measurements seem to be higher than that in the timing measurements leading to occasionally unrealistic power numbers. The third issue arises when the number of samples is not enough to estimate the shape of the frontier well.

To deal with the first two issues, we disregard samples if either decreasing values are found or if power readings differ in more than 10x between the three samples and the coordinator transitions to 3.5 GHz. While other default actions are possible, we choose to penalize ourselves when we are not confident about the interpolation. On average (geometric mean) less than 2% of samples

are discarded, but the frequency can occasionally be high, e.g., 10% for bwaves in **R(1)**. We do not correct for the third issue and our results will be suboptimal for non-convex frontiers.



(a) The Top line shows Performance-per-Watt of the best **U** mode frequency relative to **P** mode



(b) The Top line shows Performance-per-Watt of **R(10)** relative to **P** mode

Fig. 7. BIPS-per-Watt on HS with different policies.

Figure 7 compares the energy efficiency (performance-per-watt) with different modes of operation. For all these workloads, the **P** mode has the lowest efficiency. The numbers at the top show maximum gains (e.g., 1.36 implies 36% gains) in energy efficiency over **P**-mode by selecting the optimal frequency in **U** or **S** modes (Figure 7a) and **R(10)** mode (Figure 7b). We observe that:

- *The potential rewards for selecting optimal configurations are significant*: The efficiency improvements were 28.6% (imagick) to 113.7% (swim) over **P** (geometric mean: 55% over **P**, 13.4% over **S**). The **O** and **P** modes are suboptimal for this metric for every workload. However, there is a performance cost. Compared to **P** mode, the most energy-efficient frequency setting for each workload resulted in a reduction of BIPS of around 12.8% (mgrid) to 45.3% (botsspar), with a geometric mean of 35.4%.
- *There is no single best static frequency setting*: The best static frequency settings for the different workloads were 0.8 GHz (swim), 1.0 GHz (applu, graph500, hpcg), 1.4 GHz (mgrid), 1.8 GHz (bt, botsspar), 2.0 GHz (ilbdc, smithwa, kdtree), 2.1 GHz (md, nab, botsalgn, fma3d), 2.3 GHz (bwaves, imagick).
- *Rapid profiling and reconfigurations are not necessary for long running workloads*: We did a sensitivity analysis with t = 1 ms, 4 ms, 10 ms, and 20 ms. The resulting performance-per-watt numbers indicate that **R(20)** (geometric mean: 48% over **P**, 8.3% over **S**), **R(10)** (geometric mean: 49% over **P**, 9% over **S**), and **R(4)** (geometric mean: 48.7% over **P**, 8.8% over **S**) improved over **R(1)** (geometric mean: 27.5% over **P**, -6.7% over **S**).

We found that many workloads exhibit long-term variation and periodic behavior in power-performance traces. For example, applu shows 34.3 sec periodicity in **P** mode and 42.3 sec with **R(10)**. graph500 exhibits 12.3 sec periodicity in **P**-mode and 15 sec with **R(10)**. We expect long training intervals that track considerable execution history to work well with such workloads.

We use quadratic interpolation for the socket predictor. A piecewise linear interpolation would be faster, but for the performance-per-watt metric, only one of the sample frequencies (0.8/2.1/3.5 GHz) would get chosen as the optimal frequency. This is because $Perf(f) = af + b$, $Pwr(f) = cf + d \implies Perf(f)/Pwr(f)$ is monotonic in f. So, the maxima will always occur among the end points of the interval. For applu, while Linear fluctuates mostly between 0.8 and 2.1 GHz, resulting in 66% improvement over **P**-mode, Quadratic selects more frequencies in between resulting in 76.4% improvement. mgrid similarly had more improvements with Quadratic than with Linear.

Doing fixed-time experiments (running for 1200 secs), as opposed to fixed-work experiments, risks getting incomparable runs. However, for these workloads, the differences are small due to stability in the average energy efficiency over long time intervals. We re-evaluated results using the same number of instructions (minimum across all policies from the fixed-time runs) for each workload. In terms of **U**-mode gains over **P**-mode, applu changed from 84% to 80% whereas botsspar changed from 47% to 52%. The geometric mean changed < 2% for all policies. All trends remained the same.

9.1 SLAee: Adding L2 Prefetch Control

Hardware prefetching on Intel x86 machines can be enabled or disabled by writing specific values to Model-Specific Registers (MSRs) [20]. All prefetchers are enabled by default. In this study we keep the DCU (L1 Data Cache) prefetchers enabled, but dynamically enable or disable the L2 prefetchers (all cores).

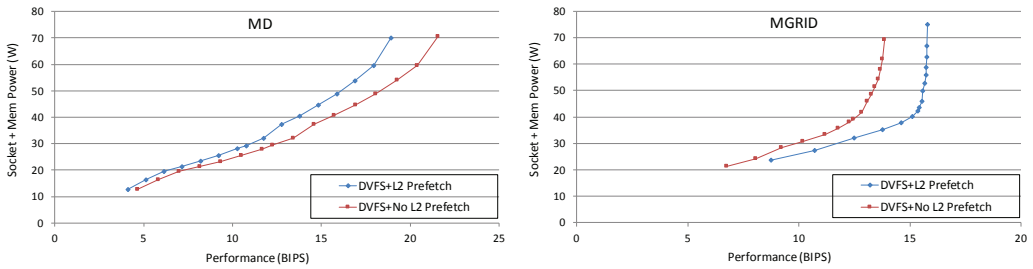


Fig. 8. Pareto-optimal states have L2 prefetching enabled for mgrid, but not for md.

Figure 8 shows one example workload each for beneficial prefetch (mgrid) and harmful prefetch (md) with all possible socket DVFS settings. When prefetching is disabled, md shows 14% improvement in peak performance and 13.6% in maximum energy efficiency whereas mgrid shows 12.3% loss in peak performance and 14.9% loss in maximum energy efficiency. Disabling prefetch also improves SPECpower performance (Figure 14). Since prefetching benefits are workload dependent, a static prefetch setting will always be suboptimal for some workloads.

We extend the frequency governor in the following simple way. Instead of taking one sample at 2.1 GHz, we take two samples—once with prefetching enabled and once disabled. We choose the prefetching mode that gives better performance and continue with that for the remaining two samples and estimating the frontier for that interval. Our choice of 2.1 GHz for taking the initial two samples is motivated by the need to keep the overhead of taking an extra sample small. A

mid-range frequency, such as 2.1 GHz, is likely to incur a lower additional overhead for this than a high frequency, such as 3.5 GHz since the energy-efficient operations for most workloads are not at high frequencies. Similar to **R(*t*)**, we name the new governor **RF(*t*)** (Reactive with prefetch control), parametrized by t, the length of the profiling interval in milliseconds. **RF(10)** improved performance-per-watt more than **R(10)** for md (48% over **P** instead of 31%) but did not create significant differences for other workloads.

To summarize, we find that the **P**, **PF** and **O** modes can always be improved by the other policies. **S** works best for swim, well for hpcg, but can be improved by **U**, **R** and **RF** for the other workloads. **U** is a good policy to use provided that workload is known in advance and profiling experiments can be carried out. **R** reaches close to **U** but is unable to outperform it. This is likely because these workloads have long-term stable behavior making the best static frequency not a bad choice. On the other hand, **R** suffers from runtime profiling overheads and prediction errors due to sampling inconsistency and non-convexity of the frontier. The situation changes for SPECpower (see Sections 11.2 and 11.3), where reactive governors do significantly better than static frequency settings. **RF** further improves upon **R** if disabling prefetch is useful but does not hurt energy efficiency if not, so it represents the best of both prefetch modes.

9.2 SLAee: Adding Control for Wall Power

Measuring wall power (to include power consumption by the PSU, network interfaces, hard disks, etc.) with external power meters causes a time granularity mismatch, e.g., minimum 1 sec intervals with Watts Up? meters as opposed to milliseconds (supported by the RAPL counters) used by our governors (**R(10)** uses a measurement interval granularity of 10 ms). So we estimate wall power from socket+mem power (measured by RAPL counters) using regression models determined offline by running workloads (SPECOMP2012, graph500, hpcg) with all frequency settings. We consider the following two models for wall power (y) as a function of socket+mem power (x).

(1) Quadratic: $y = ax^2 + bx + c$. The best-fit line is $y = 0.004x^2 + 0.7932x + 21.329$ with coefficient of determination $R^2 = 0.9912$.
(2) Linear: $y = ax + b$. The best-fit line is $y = 1.141x + 14.891$ with $R^2 = 0.9869$.

The quadratic model gives a better fit for idle power as well as a slightly better fit overall, so we use this model for subsequent experiments.

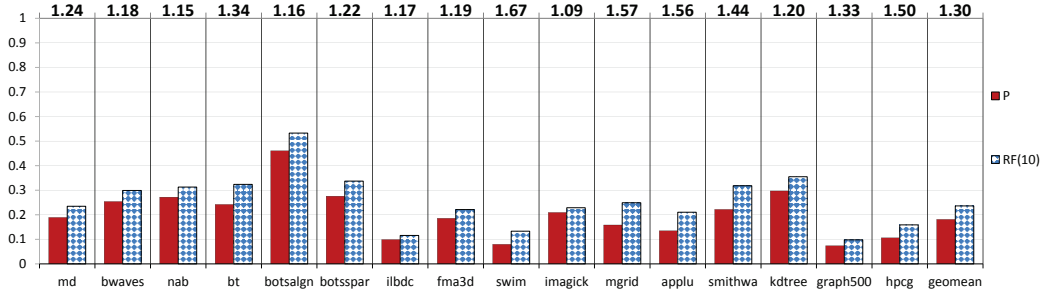


Fig. 9. BIPS-per-watt of governors **P** and **RF**(10) with wall (full-system) power.

Figure 9 shows the performance-per-watt for the baseline **P** governor (measured wall power) and our new **RF** governor. **RF** reads RAPL counters, then applies the wall power model mentioned above to estimate wall power. The maximum gains in performance-per-watt achieved by **RF** over **P** is 67% (swim) while the geometric mean over all workloads is 30.2%. Since CPUE is

directly proportional to energy consumption and inversely proportional to energy efficiency, we have $\frac{CPUE(\mathbf{P})}{CPUE(\mathbf{RF})} = \frac{E(\mathbf{P})}{E(\mathbf{RF})} = \frac{\eta(\mathbf{RF})}{\eta(\mathbf{P})} = 1.302$. So the average energy savings of **RF** over **P** is $\frac{E(\mathbf{P})-E(\mathbf{RF})}{E(\mathbf{P})} = 1 - \frac{E(\mathbf{R}F)}{E(\mathbf{P})} = 1 - 1.302^{-1} = 23.2\%$. **RF** can thus save cost by using 23% less energy on average to do the same work or generate revenue by doing 30% more work on average for the same energy cost. While it improves energy-efficiency, **RF** loses performance, with respect to **P** mode, ranging from 0.2% (md) to 30% (bt) with a geometric mean of 19.5%.

10 SLAPOWER: MAXIMIZE PERFORMANCE WITHIN A POWER CAP/BUDGET

None of the standard Linux governors **S**, **C**, **O**, **U**, **P** deal with power caps/limits. The RAPL [21] capabilities include mechanisms to enforce a limit on the power consumption. One advantage of RAPL limits over frequency settings is that they can be fine-grained (e.g., units of 1/8 W) leading to greater control of the state space. Another advantage is that since RAPL limits are enforced by the hardware, the management overhead is lower than that of a software-controlled governor. Prior works [26, 39] have used power limiting as a mechanism to improve energy efficiency.

There are two main disadvantages of the RAPL power-capping mechanisms. First, capping of wall power cannot be directly specified. Second, non-frequency settings (e.g., prefetching) are not managed by the RAPL mechanisms. So, workloads such as md that benefit significantly from prefetch control would not see those advantages with the RAPL approach. From this perspective, RAPL guarantees a power cap, *not best performance within that power cap.* Pareto optimality provides the stronger guarantee.

Our new governor for SLApower, **RF_SLApower(*t*)**, improves upon the RAPL governor in both of these aspects. It controls both the prefetch settings as well as socket frequency to get the maximum performance within a specified power budget. We modify the objective selector to select the next state that is predicted to use the highest power among all states with less power consumption than the SLA target. We do not set turbo mode frequencies because of limited control in turbo mode, so our maximum frequency is the nominal frequency (3.5 GHz).

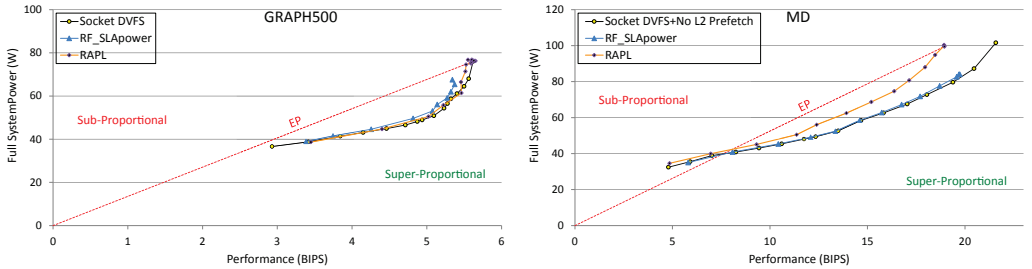


Fig. 10. Power-performance profiles for graph500 and md for SLApower.

We investigate enforcement of SLApower using **RF_SLApower(10)** and the RAPL governor for two of our workloads: graph500 and md. For RAPL, we limit average socket power over 1 second intervals from 10W to 80W in steps of 5W. We could enforce a power cap only for the socket on HS, not for the memory or for other components. We used caps on full system power for **RF_SLApower(10)** (35W–80W for graph500 and 35W–105W for md, both in steps of 5W) and a more restrictive interval of 510 ms.

Figure 10 shows the power-performance profiles for the governors. Both workloads exhibited behavior close to Pareto optimal with **RF_SLApower(10)**. The RAPL governor works well for

graph500, but causes Pareto-dominated (hence suboptimal) operations for md as it does not control prefetch settings. **RF_SLApower(10)** falls short of achieving maximum performance at the upper end of the operating range due to not operating in the turbo region.

11 SLAPERF: MAXIMIZE POWER SAVINGS GIVEN A PERFORMANCE TARGET

None of the standard governors **S**, **C**, **O**, **U**, **P** deal with user-specified performance targets. Our new governor, **RF_SLAperf(*t*)**, allows the user to specify performance targets in absolute or relative (with respect to peak) BIPS.

11.1 Governing for absolute performance targets

To govern for this SLA, we keep the Pareto predictor intact, but modify the objective selector to keep track of the performance so far (time elapsed and instructions executed). The idea is to determine the desired performance for the next interval so that if this target is met, then the average performance so far would be that required by the SLA. If the average performance so far is greater than the SLA, the lowest performing point is chosen. Otherwise, if the next interval target is greater than the best performance predicted for 3.5 GHz, then turbo mode is chosen. Otherwise, the point on the frontier that meets or just exceeds the next interval target is chosen.

We investigate enforcement of SLAperf using **RF_SLAperf(10)** for two of our workloads: md and graph500. md has mostly homogeneous behavior during its execution and we will show that it can be governed well to meet the SLA. On the other hand, graph500 has significant heterogeneity (different execution phases) and, as we shall show, cannot be governed well without prior knowledge of the phase behavior.

md has an average performance range of 4.8–21.6 BIPS at the frontier depending on the frequency and with L2 prefetching disabled. We select SLA targets of 5.0, 7.5, 10.0, 12.5, 15.0, 17.5, 20.0 and 22.5 BIPS (unreachable). graph500 has an average performance range of 2.9–5.6 BIPS at the frontier depending on the frequency and with L2 prefetching enabled. We select SLA targets of 3.0, 3.5, 4.0, 4.5, 5.0 and 5.5 BIPS.

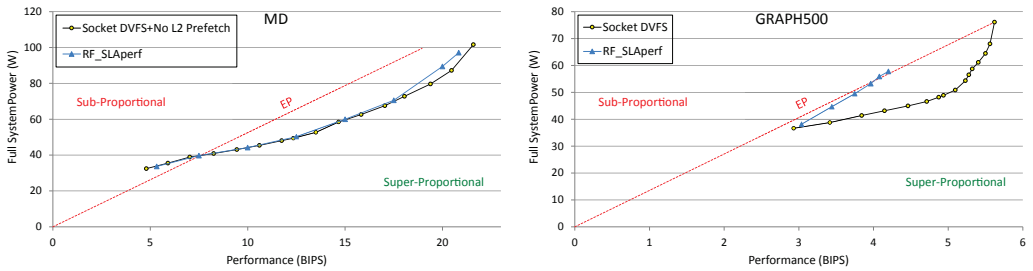


Fig. 11. Power-performance profiles for md and graph500 for SLAperf.

Figure 11 shows the power-performance profiles and expected behavior for both workloads. The profile for md was at the frontier except at high performance targets. Its highest performance is around 3.5% less than the maximum possible due to sampling/profiling and prediction overheads in the governor. For targets close to its highest performance, it tries to compensate for the loss by transitioning more into turbo mode resulting in more power consumption and consequently, Pareto-dominated states.

The governor failed to meet the SLA for most points of graph500 and the profile was quite suboptimal, being closer to proportional than Pareto optimal. However, as we explain below,

this is primarily due to the non-homogeneous nature of the workload rather than incorrect state transitions chosen by the governor.

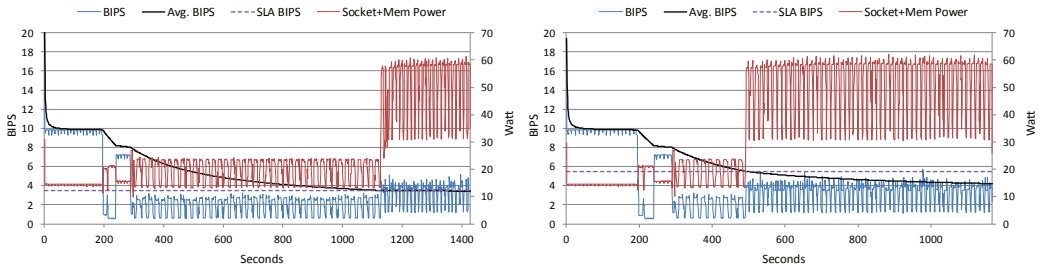


Fig. 12. Execution traces for graph500 with SLA = 3.5 BIPS and 5.5 BIPS.

Figure 12 shows execution traces for graph500 for two SLAs: 3.5 and 5.5 BIPS. The primary vertical axis (left axis) and the blue line show the number of instructions executed per second in BIPS. The secondary vertical axis (right axis) and the red line show the power (socket+memory) in Watts. The Avg. BIPS line shows the average BIPS attained by the workload execution so far. The dashed line shows the average BIPS expected to be reached over the entire execution. The governor seeks state transitions that will maintain the average BIPS equal to the SLA BIPS.

graph500 has non-homogeneous behavior—the initial approximately 290 seconds is mostly a high-performance phase whereas the remainder of the execution (successive iterations of breadth-first search) is low-performance. Initially, the average BIPS is higher than the SLA BIPS, so the governor reduces frequency to the lowest possible to save power. Eventually, execution enters the low-performance phase and the average BIPS starts dropping more rapidly. However, the governor continues with a low frequency execution as the SLA BIPS is still lower than the average BIPS. After a while (1152 seconds for SLA=3.5, 491 seconds for SLA=5.5), the average drops below the SLA BIPS and remains that way although the governor transitions to higher frequencies including turbo mode (with a sharp increase in power consumption). This is more pronounced for SLA=5.5, where the SLA is breached earlier, than for SLA=3.5. Timely prediction of maximum performance in future execution phases is needed to handle this issue.

11.2 Governing for relative performance targets

Figure 13 shows power-performance profiles for SPECpower with the default governors. **P**-mode and **O**-mode perform similarly and use the highest power for the achieved load. **S**-mode uses the lowest power for the achieved load, but the maximum load achievable is low (see below). **C**-mode works better than **P**-mode or **O**-mode at low loads but in general consumes significantly more power than **S**-mode or the **U**-mode Pareto frontier for the same load. For these experiments we "niced" the power measurement daemon and set the ignore_nice_load parameter of the **O** and **C** governors to discount activity by the daemon while calculating processor utilization.

Even though **S**-mode's profile lies on the Pareto frontier, it has two limitations. First, it can only serve up to around 27% of peak load. So it will fail the performance requirement (see Section 2) at higher loads. The other governors in Figure 13 can serve high as well as low loads. Second, although RUE = 1 for this governor, LUE $\geq$ 1.58. So, at least 58% excess energy is used compared to operating with the best load even though it is operating at the Pareto frontier. Its profile does not even enter the Super-Proportional region. So, restricting servers to this policy is not a good

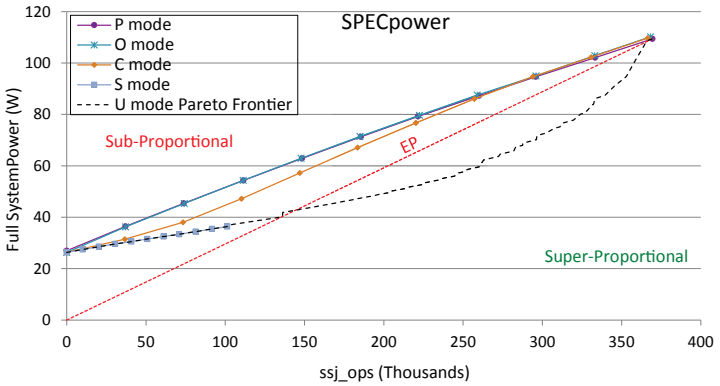


Fig. 13. Power-performance profiles for SPECpower with Linux governors.

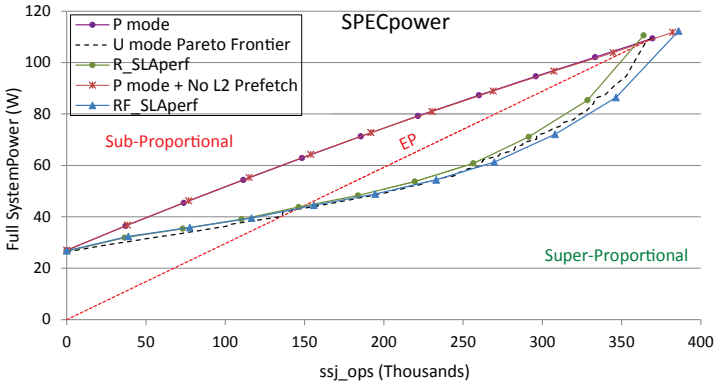
idea. The other governor profiles are distant from Dynamic EO, thus have large RUE values (waste energy).

To govern for this SLA, we modify the Pareto predictor to also sample turbo mode so that peak BIPS can be estimated. We now have the maximum profiling frequency as 3.7 GHz (midpoint of the turbo range 3.5–3.9 GHz) since we do not know the actual temperature-dependent operating frequency. For any load, the target relative BIPS is set to be equal to that load level (load relative to maximum load). For example, attempting to serve 70% load level sets the target relative BIPS to 0.7. The objective selector selects the lowest-power configuration that is predicted to have relative BIPS greater than or equal to the target relative BIPS.

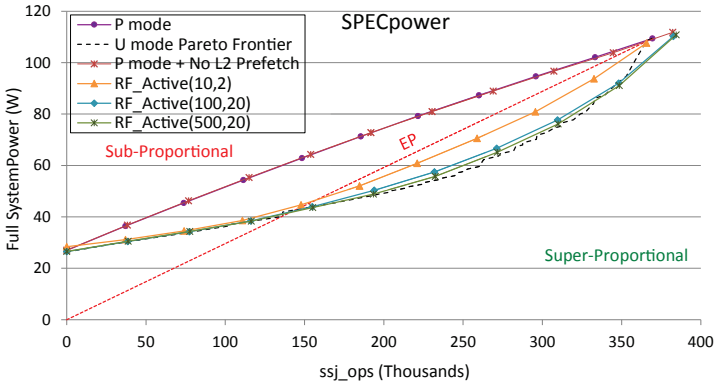
We call this new governor **RF_SLAperf(*t*)**. The parameter t, in milliseconds, is the length of the intervals as follows. For 100% target performance, the plan is to turn prefetching off and on for t ms each, then choose the mode that performed better to run with for the next $48t$ ms. This plan is then repeated. For 100% target performance, the frequency is kept at 3.7 GHz and not changed. For a lower target performance, the plan is to set frequency to 2.1 GHz (midpoint frequency) for profiling prefetch settings (t ms for each mode). With the chosen prefetching mode, set frequency to 3.7 GHz (turbo frequency) and then to 0.8 GHz (lowest frequency) for t ms each. We then estimate the Pareto frontier, select the best frequency, and run with that setting for the next $48t$ ms.

Figure 14a shows power-performance with **R_SLAperf(10)** and **RF_SLAperf(10)**. **R_SLAperf(*t*)** always enables prefetching whereas **RF_SLAperf(*t*)** dynamically controls it. Controlling prefetching increases the maximum load achievable compared to **P**-mode that always has prefetching enabled. **RF_SLAperf(10)** outperforms **P**-mode by around 4%. This outcome is qualitatively similar to md (See Figure 10). The increase in maximum performance can also be observed by running in **P**-mode with prefetching disabled (**P**-mode + No L2 Prefetch profile).

RF_SLAperf(10) disabled prefetching for most of the time and chose a number of different frequencies for each run depending on the load served. By default, SPECpower does 3 calibration intervals followed by 11 measurement intervals. During the calibration intervals and the first measurement interval, the server is offered very high load. So we expect that during these times the maximum frequency would be chosen by the governor. Thus for at least (3+1)/(3+11) = 28.6% (observed: 34%) of the time, the maximum frequency should be chosen. The last measurement interval is "Active Idle", that is zero load is offered, so we expect to select the lowest frequency during this interval which is around 1/(3+11) = 7.1% (observed: 15%) of total time.



(a)



(b)

Fig. 14. Power-performance with (a) **RF_SLAperf**, **R_SLAperf**, and (b) **RF_Active**.

11.3 Governing to minimize idle time

So far, a performance target (absolute or relative) needed to be specified to the governors so that they could ensure that SLAperf is satisfied. We will now discuss a new governor that aims to serve the offered load without keeping any threads idle. We demonstrate its action in the context of the SPECpower workload execution.

The key idea is to predict the highest frequency such that there are no idle cycles. Let α denote the number of active (that is, not idle) cycles per second in the last interval. The governor estimates the optimal value of target frequency for the next interval to be $\frac{\alpha}{8}$. The division by 8 is done since there are 8 logical threads on HS. This assumes that in the next interval, the load will remain the same (or at least, not increase) and all threads will be equally active.

In case these assumptions are not true, the system may not be able to serve the offered load. To protect against this situation, the governor increases the estimated target frequency by a step whenever it equals the current frequency and doubles the value of the step. This facilitates an exponential ramp-up of frequency over successive intervals. On the other hand, if the estimated

target frequency is less than the current frequency, the frequency is set to the target frequency and the step is re-initialized.

We call our new governor **RF_Active(*t*,*p*)**. It repeatedly does the following. It turns prefetching off and on for $p/2$ ms each. Then it chooses the prefetching mode that performed better and also estimates and sets the frequency for the remaining interval. We determine the length of the last interval offline so that the average total epoch length is t ms including governor overheads (system calls, profiling, and estimations). This governor does not predict either power or performance for any configuration, but manages to operate the system very close to Dynamic EO.

Figure 14b shows the power-performance profile of our new governor, **RF_Active**, with different parameter values: **RF_Active(10,2)**, **RF_Active(100,20)**, and **RF_Active(500,20)**. For these experiments, we consider all frequencies (0.1 GHz granularity). **RF_Active(10,2)** suffers from more overheads and less accurate selection of resource settings with shorter intervals—it keeps prefetching enabled for a larger fraction of time and also selects the maximum frequency more often than **RF_Active(100,20)** or **RF_Active(500,20)**. Compared to **RF_SLAperf(10)**, **RF_Active(100,20)** and **RF_Active(500,20)** select the lowest frequency more often and the maximum frequency less often but keep prefetching enabled for longer.

RF_Active is similar to PAMPA [42] since both governors change core frequency in response to core utilization. The description of PAMPA does not include the rate of frequency change in each step, but we believe that our exponential ramp-up strategy allows **RF_Active** to increase service rates faster in response to a sudden increase in load levels. PAMPA controls the number of active cores in addition to core (socket) frequency but does not control cache prefetch enable. Number of active cores as well as thread placement [41] are interesting resource management knobs that can enhance the potentials for power savings.

12 LIMITATIONS

We will now discuss a few limitations of our governor designs—socket-wide control, intrusive profiling, sampling inconsistency, non-zero reaction times, and not tracking latency distributions.

Socket-Wide Control: Our governors select the same resource setting for the entire socket. Having the same frequency setting for all cores may be suboptimal for non-homogeneous workloads if the hardware supports different settings for different cores. We expect that we can continue to profile with all-core settings and then predict per-core Pareto frontiers. To do that we need to measure or estimate, using performance counter values, the performance and energy consumption of each core for each profiled setting. But interpolating Pareto frontiers individually for many cores can be costly in software. Hardware support for doing this would be useful.

Intrusive Profiling: Our governors try out a few resource configurations (e.g., socket frequencies) to determine their effectiveness. This intrusive profiling may be costly in terms of reconfiguration latency and energy for some resources, e.g., caches. Moreover, exhaustively profiling a multitude of resources can be prohibitively expensive for short time intervals. To limit overheads, profiling can be combined with other strategies, such as analytical models and heuristics, to guide resource configuration decisions. For example, thread criticality can be used to partition aggregate frequency among cores [27]. CoScale [10] uses analytical performance and energy models parametrized by profiled values of performance counters and activity counters at the highest core and memory frequencies.

Sampling Inconsistency: The strategy of sampling execution with three frequencies and then fitting a quadratic polynomial to the measured power-performance values assumes that the samples are consistent, that is, not drawn from different execution phases. Sample inconsistency can be avoided by an alternate approach [34, 38] that samples performance counters once, then predicts

power and performance for other configurations using a precharacterized model. However, that approach is limited by the number of performance counters that can be concurrently read (can affect prediction accuracy) and by the availability of the particular counters on different platforms (can affect portability).

Non-Zero Reaction Times: Our **R(*t*)** governors logically partition execution time into epochs, of length $51t$ or $52t$. Shortening the epochs will allow for faster reaction times with increased profiling overhead. However, they need to be long enough to capture a stable, representative region of workload execution. Moreover, the unit of time, t, cannot be made very small due to limited update frequency of the RAPL counters (usually about once every millisecond [15, 21]).

Latency Distribution Unawareness: Our governors track performance in terms of throughput metrics but not in terms of distributions of computation latency/response times. However, some online applications such as web services may have requirements on tail latency that need to be met [8]. Recent works [17, 23, 24, 26, 29, 30] have explored latency-aware resource management (without prefetching control). We plan to augment our governors to be aware of latency requirements, possibly with application-specific knowledge, as part of future work.

13 RELATED WORK

Hsu and Poole [16] observed real machines doing better than the conventional "ideal" system that assumes linear proportionality. They proposed quadratic proportionality ($\text{Power}(u) \propto u^2$, where u is the load level) as the new ideal model. However, this makes ideal system efficiency load-dependent $\left(\frac{u}{\text{Power}(u)} = \frac{1}{u}\right)$, with higher efficiency at lower loads than at higher loads. Wong also observed that the peak efficiency can occur at intermediate loads, in the super-proportional region [44].

Wong and Annavaram [45] name the region that we call Sub-Linear as Superlinear. We prefer to use "Sub-" in the sense that operating in this region lowers efficiency compared to that of Linear (Dynamic EP).

Song et al. [35] proposed Iso-energy-efficiency (EE) as the energy ratio between sequential and parallel executions of a given application. Our CPUE, LUE and RUE metrics do not use specific execution modes (e.g., sequential/parallel, homogeneous/heterogeneous, speculative/non-speculative, cache-conscious/cache-oblivious, etc.) for reference, but compare system states to the Pareto frontier (Dynamic EO) or to EOP. The definitions of our metrics are oblivious to which configurations created the frontier. The EE model focuses on maintaining equal efficiency as systems and applications scale up. In contrast, the EP and EOP models focus on maintaining equal efficiency under changing loads. EE does not quantify its dependence on load.

Barroso and Hölzle [4] compute datacenter energy consumption as PUE × SPUE × energy to electronic components. While PUE accounts for non-compute overheads in datacenter building infrastructure, SPUE (Server PUE) accounts for overheads, e.g., power supply losses, to computing energy. Our RUE and LUE metrics do not separate SPUE losses from computing energy but separate energy-wasting operating configurations and loads from optimal ones.

Other existing metrics [32, 40, 45] for characterizing energy efficiency, e.g., based on the dynamic power range, ratio between the idle and peak power consumptions, deviations from or area enclosed by an ideal curve, etc. continue to be useful with the new ideals, EOP and Dynamic EO, replacing the conventional ideals.

Dynamic voltage and frequency scaling (DVFS) and dynamic frequency scaling (DFS) for cores are well-known techniques [13, 18, 43]. Researchers have explored mechanisms [10, 22, 24, 27, 30] to exploit per-core DVFS capabilities. Our server, HS, lacks this support (also see Section 12).

OS governors [6, 34, 38] typically control DVFS settings but not additionally prefetch settings.

Seeking Pareto optimality for improving efficiency is well-known [2, 5, 25, 31, 44], but its relationship to energy proportionality had not been formalized (see Section 3 for invariants between Dynamic EP, EP, Dynamic EO and EOP).

14 CONCLUSION

Power-performance Pareto optimality and energy proportionality are well-known but dissimilar concepts, both of which share the end goal of making computing more energy efficient. We demonstrated that the conventional model of energy proportionality is inadequate for reconfigurable systems since it does not guarantee energy optimality. We defined a new model, EOP, that guarantees both optimality and proportionality and established its relation to the Pareto frontier.

We proposed a new metric, Computational PUE (CPUE), that quantifies how much excess computational energy is used by real systems relative to that by EOP. This depends on both the load served and the system configuration used to serve that load. Our new Iron Law of Energy shows how CPUE can be decomposed into constituent terms that help separate the load, configuration, and design aspects of suboptimality.

We developed new SLA-aware OS governors that seek Pareto optimality, and thereby reduce CPUE, in the presence of frequency scaling and cache prefetching. We demonstrated improvements in performance-per-watt by up to 67% (maximum gains) and 30% on average (geometric mean over all workloads) of a modern Intel Haswell server machine. This opens up significant opportunities for revenue generation or cost savings. We also presented case studies and discussed challenges in governing for maximizing performance within a power cap and minimizing power for a performance target. Scheduling frameworks that carefully choose configurations and operating ranges will unlock the full potential of current and future reconfigurable systems.

ACKNOWLEDGMENTS

We thank Luiz Barroso, Edouard Bugnion, Mark Hill, Michael Marty, Kathryn McKinley, Michael Swift, and anonymous reviewers for helpful comments. This work was supported in part by the National Science Foundation, under grant CCF-1218323, grant CNS-1302260, and grant CCF-1533885. The views expressed herein are not necessarily those of the NSF. Wood has significant financial interests in AMD and Google.

REFERENCES

[1] Victor Avelar, Dan Azevedo, and Alan French (Eds.). 2012. *PUE: A Comprehensive Examination of the Metric.* The Green Grid.

[2] Peter E. Bailey, Aniruddha Marathe, David K. Lowenthal, Barry Rountree, and Martin Schulz. 2015. Finding the Limits of Power-constrained Application Performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15)*. ACM, New York, NY, USA, Article 79, 12 pages.

[3] Luiz André Barroso and Urs Hölzle. 2007. The Case for Energy-Proportional Computing. *Computer* 40, 12 (Dec. 2007), 33–37.

[4] Luiz André Barroso and Urs Hölzle. 2009. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines* (first ed.). Morgan & Claypool Publishers.

[5] Luca Benini, Alessandro Bogliolo, Giuseppe A. Paleologo, and Giovanni De Micheli. 1999. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18, 6 (Jun 1999), 813–833.

[6] Dominik Brodowski and Nico Golde. 2015. CPU frequency and voltage scaling code in the Linux(TM) kernel. Linux CPUFreq. CPUFreq Governors. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt. (2015).

[7] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. 2001. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP '01)*. ACM, New York, NY, USA, 103–116.

[8] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (Feb. 2013), 74–80.

[9] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. ACM, New York, NY, USA, 127–144.

[10] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. 2012. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*. IEEE Computer Society, Washington, DC, USA, 143–154.

[11] Jack Dongarra and Michael A. Heroux. 2013. *Toward a New Metric for Ranking High Performance Computing Systems*. Technical Report SAND2013-4744. Sandia National Laboratories.

[12] Electronic Educational Devices Inc. 20xx. Watts Up? .Net. https://www.wattsupmeters.com/secure/products.php?pn=0&wai=0&spec=2. (20xx).

[13] Kinshuk Govil, Edwin Chan, and Hal Wasserman. 1995. Comparing Algorithm for Dynamic Speed-setting of a Low-power CPU. In *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking (MobiCom '95)*. ACM, New York, NY, USA, 13–25.

[14] Graph 500. 2011. Graph 500 Reference Implementation v2.1.4. http://www.graph500.org/referencecode. (2011).

[15] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. 2012. Measuring Energy Consumption for Short Code Paths Using RAPL. *SIGMETRICS Performance Evaluation Review* 40, 3 (Jan. 2012), 13–17.

[16] Chung-Hsing Hsu and S.W. Poole. 2013. Revisiting Server Energy Proportionality. In *2013 42nd International Conference on Parallel Processing (ICPP)*. 834–840.

[17] C. H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski. 2015. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 271–282.

[18] Intel Corporation. 2004. Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor. (March 2004).

[19] Intel Corporation. 2012. Intel Performance Counter Monitor - A better way to measure CPU utilization. https://software.intel.com/en-us/articles/intel-performance-counter-monitor. (2012).

[20] Intel Corporation. 2014. Disclosure of H/W prefetcher control on some Intel processors. https://software.intel.com/en-us/articles/disclosure-of-hw-prefetcher-control-on-some-intel-processors. (September 2014).

[21] Intel Corporation. 2015. Intel 64 and IA-32 Architectures Software Developer's Manual, Vol. 3B: System Programming Guide, Part 2. (2015).

[22] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. 2006. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 39)*. IEEE Computer Society, Washington, DC, USA, 347–358.

[23] Svilen Kanev, Kim Hazelwood, Gu-Yeon Wei, and David Brooks. 2014. Tradeoffs between Power Management and Tail Latency in Warehouse-Scale Applications. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 31–40.

[24] Harshad Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast Analytical Power Management for Latency-critical Systems. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. ACM, New York, NY, USA, 598–610.

[25] J. Li and J.F. Martinez. 2006. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *The Twelfth International Symposium on High-Performance Computer Architecture*. 77–87.

[26] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards Energy Proportionality for Large-scale Latency-critical Workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA '14)*. IEEE Press, Piscataway, NJ, USA, 301–312.

[27] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. 2011. Scalable Power Control for Many-core Architectures Running Multi-threaded Applications. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*. ACM, New York, NY, USA, 449–460.

[28] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. ACM, New York, NY, USA, 248–259.

[29] David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. 2011. Power Management of Online Data-intensive Services. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*. ACM, New York, NY, USA, 319–330.

[30] George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. 2015. Energy Proportionality and Workload Consolidation for Latency-critical Applications. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15)*. ACM, New York, NY, USA, 342–355.

[31] Martino Ruggiero, Andrea Acquaviva, Davide Bertozzi, and Luca Benini. 2005. Application-Specific Power-Aware Workload Allocation for Voltage Scalable MPSoC Platforms. *International Conference on Computer Design (ICCD)*

(2005), 87–93.
[32] Frederick Ryckbosch, Stijn Polfliet, and Lieven Eeckhout. 2011. Trends in Server Energy Proportionality. *Computer* 44, 9 (Sept. 2011), 69–72.
[33] Sandia National Laboratories. 2014. High Performance Conjugate Gradients v2.1. https://software.sandia.gov/hpcg/download.php. (Jan. 2014).
[34] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. 2009. Koala: A Platform for OS-level Power Management. In *Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys '09)*. ACM, New York, NY, USA, 289–302.
[35] S. Song, C. Y. Su, R. Ge, A. Vishnu, and K. W. Cameron. 2011. Iso-Energy-Efficiency: An Approach to Power-Constrained Parallel Computation. In *IEEE International Parallel Distributed Processing Symposium (IPDPS)*. 128–139.
[36] SPEC. 2008. SPECpower_ssj. https://www.spec.org/power_ssj2008/. (2008).
[37] SPEC. 2012. SPEC OMP2012. http://www.spec.org/omp2012/. (2012).
[38] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. 2011. Green governors: A framework for Continuously Adaptive DVFS. In *International Green Computing Conference and Workshops (IGCC)*. 1–8.
[39] Balaji Subramaniam and Wu-chun Feng. 2013. Towards Energy-proportional Computing for Enterprise-class Server Workloads. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13)*. ACM, New York, NY, USA, 15–26.
[40] Georgios Varsamopoulos and Sandeep K. S. Gupta. 2010. Energy Proportionality and the Future: Metrics and Directions. In *39th International Conference on Parallel Processing Workshops (ICPPW)*. 461–467.
[41] Augusto Vega, Alper Buyuktosunoglu, and Pradip Bose. 2013. SMT-centric Power-aware Thread Placement in Chip Multiprocessors. In *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques (PACT '13)*. IEEE Press, Piscataway, NJ, USA, 167–176.
[42] Augusto Vega, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, and Srinivasan Ramani. 2013. Crank It Up or Dial It Down: Coordinated Multiprocessor Frequency and Folding Control. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, USA, 210–221.
[43] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. 1994. Scheduling for Reduced CPU Energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation (OSDI '94)*. USENIX Association, Berkeley, CA, USA, Article 2.
[44] Daniel Wong. 2016. Peak Efficiency Aware Scheduling for Highly Energy Proportional Servers. In *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA '16)*.
[45] Daniel Wong and Murali Annavaram. 2012. KnightShift: Scaling the Energy Proportionality Wall Through Server-Level Heterogeneity. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*. IEEE Computer Society, Washington, DC, USA, 119–130.

Received August 2016; revised December 2016; accepted January 2017