JouleGuard: Energy Guarantees for Approximate Applications

Henry Hoffmann
University of Chicago, Department of Computer Science
hankhoffmann@cs.uchicago.edu

Abstract

Energy consumption limits battery life in mobile devices and increases costs for servers and data centers. *Approximate computing* addresses energy concerns by allowing applications to trade accuracy for decreased energy consumption. Approximation frameworks can guarantee accuracy or performance and generally minimize energy; however, they provide no energy guarantees. Such guarantees would be beneficial for users who have a fixed energy budget and want to maximize application accuracy within that budget. We address this need by presenting JouleGuard: a runtime control system that coordinates approximate applications with system resource usage to provide control theoretic formal guarantees of energy consumption, while maximizing accuracy. We implement JouleGuard and test it on three different platforms (a mobile, tablet, and server) with eight different approximate applications created from two different frameworks. We find that JouleGuard respects energy budgets, provides near optimal accuracy, adapts to phases in application workload, and provides better outcomes than application approximation or system resource adaptation alone. JouleGuard is general with respect to the applications and systems it controls, making it a suitable runtime for a number of approximate computing frameworks.

Categories and Subject Descriptors D.4.8 [*Operating Systems*]: Performance—Measurements, Monitors; I.2.8 [*Problem Solving, Control Methods, Search*]: Control Theory

General Terms Design, Experimentation, Measurement, Performance

Keywords Adaptive software, control theory, dynamic systems

SOSP'15, October 4–7, 2015, Monterey, CA.

http://dx.doi.org/10.1145/2815400.2815403

1. Introduction

Energy consumption is crucial to the full spectrum of computing systems from mobile, where it determines battery life, to supercomputing, where it affects costs. Energy usage has been confronted at the application-level through *approximate* computing, which trades accuracy for reduced resource usage (see Sec. 6.1 for examples). At the system-level, energy has been confronted through *energy-aware* systems that trade performance for reduced resource usage (see Sec. 6.2 for examples). *Cross-layer* approaches coordinate application and system (see Sec. 6.3 for examples).

The combination of approximate applications and energy-aware systems creates a huge design space characterized by three features: 1) the level at which tradeoffs are made (application, system, or both), 2) the properties traded (accuracy, performance, energy), and 3) the objectives (to minimize/maximize versus provide guarantees). For example, Green [5] and EnerJ [53] are application-level approaches that guarantee accuracy while minimizing energy consumption. Several resource managers provide performance guarantees (for real-time or QoS) while minimizing energy consumption [10, 23, 26, 27, 37, 43, 64]. Cross-layer approaches coordinate both application and system to achieve performance guarantees with greater energy savings than can be achieved by either alone [33, 36, 63].

One critical point currently unaddressed in this design space is *providing energy guarantees while maximizing accuracy*. Such an approach would help the many users who do not need minimal energy, but instead want a guarantee that energy consumption will not hit a critical threshold. For example, few mobile users want to minimize energy – they need guarantees that their battery will last until they return to a charger. Ideally, the system would maximize accuracy (*i.e.,* user-experience) while ensuring users only run out of charge at the very time they reach their charger. Energy guarantees are analogous to real-time and quality-of-service constraints. When meeting such timing guarantees, the OS does not maximize performance, but instead schedules jobs to complete at the deadline, ensuring the timing and minimizing the impact on the rest of the system [61]. Similarly, providing energy guarantees for approximate computations should minimize the effect on the result's accuracy.

1.1 Energy Guarantees with JouleGuard

This paper meets the challenge of providing energy guarantees for approximate applications by presenting JouleGuard. JouleGuard takes an energy goal and dynamically configures application and system to ensure that the goal is met and application accuracy is near optimal. *The key insight is that this complicated multidimensional optimization problem can be split into two sub-problems: maximizing system energy efficiency and dynamically managing application accuracy/performance tradeoffs.* Energy can be affected by making the system more efficient or speeding up the application. Thus, JouleGuard first adjusts resource usage to maximize energy efficiency. Any additional speedup required to meet the energy goal comes from carefully tuning application accuracy. Because both sub-problems may alter performance, they are dependent – each affects the other. JouleGuard, however, provides formal, control-theoretic [17], guarantees that the energy goal will be respected, despite the dependence. These guarantees come from a novel combination of machine learning and control theoretic techniques. JouleGuard is general with respect to both the approximate applications and energy-aware systems it coordinates, making it compatible with multiple approximate computing approaches and system resources.

1.2 Summary of Results

We implement JouleGuard in C and test it on three hardware platforms (a heterogeneous mobile processor, a tablet, and a server) running eight different approximate applications (five built with PowerDial [25] and three built with Loop Perforation[56]). Our results show:

- **Low Overhead:** JouleGuard adds only small overhead, in terms of both energy and performance. (See Sec. 5.1).
- **Stability and Convergence:** JouleGuard quickly converges to a given energy goal with low error. On average, across all applications, all machines, and a number of energy goals, JouleGuard maintains energy within a few percent of the goal. (See Sec. 5.3.)
- **Optimality:** JouleGuard converges to the energy goals with near optimal accuracy. On average, for all applications, systems and goals, JouleGuard is within a few percent of true optimal accuracy (see Sec. 5.4.) JouleGuard provides greater accuracy than the best that could be achieved through application approximation or system resource allocation alone (see Sec. 5.5.)
- **Responsiveness:** JouleGuard quickly reacts to application phases, automatically increasing accuracy whenever the application workload allows. (See Sec. 5.6.)

1.3 Contributions

This paper makes the following contributions:

- It presents a case for providing energy guarantees.
- Demonstration that maximizing accuracy on an energy budget can be split into two sub-problems: 1) finding the most energy-efficient system configuration and 2) tuning application performance to provide additional energy savings while maximizing accuracy.
- A machine-learning solution to maximizing energy efficiency (Sec. 3.2) and a control theoretic solution to managing application performance (Sec. 3.3).
- Formal analysis that the runtime is robust and converges to the energy goals despite the inherent dependence between application and system (Sec. 3.4).
- Empirical evaluation of JouleGuard on multiple real systems with different applications (Sec. 5).

To the best of our knowledge, this is the first cross-layer approach that provides formal guarantees of energy consumption while maximizing accuracy. The rest of this paper is organized as follows. Sec. 2 presents an example of maximizing search accuracy on an energy budget. Sec. 3 discusses JouleGuard's design and formal guarantees. Sec. 4 describes the applications and systems used to evaluate JouleGuard in Sec. 5. Sec. 6 compares JouleGuard to existing work in approximate applications, energy-aware systems, and cross-layer approaches. Sec. 7 concludes.

2. Motivation

This section demonstrates the challenges of maximizing accuracy on an energy budget and builds intuition for the formal analysis to follow. We consider document search using an approximate version of the open-source swish++ search engine [60], which varies the number of documents it returns per query [25]. We run the search engine on a server platform which supports several configurations controlling tradeoffs between power and performance.

We configure swish++ as a web server, and deploy it on our system with 32 threads[1]. We measure search performance and the total system power consumption with the application in its default (full accuracy) configuration using all system resources (see Sec. 4 for a detailed description of the system). In this default configuration, swish++ processes 3100 queries per second (qps) at an average power consumption of 280 Watts, or 0.09 Joules per query.

For this example, we want to reduce energy consumption by $1/3$ to 0.06 Joules per query (the full evaluation tests a wide range of energy goals). We could achieve this energy with a 50% performance increase, a 33% power decrease, or by some combination of the two. The primary challenge is determining which combination will result in the highest application accuracy.

We are not aware of any existing system that provides overall system energy guarantees while maximizing accuracy. It is tempting, then, to start with existing approaches

[1] We use public domain books from Project Gutenberg as our search documents. For search queries, we construct a dictionary of all words present in the documents, excluding stop words, and select words at random following a power law distribution. This is the same experimental setup used in prior work to test a variable accuracy version of swish++ [25].

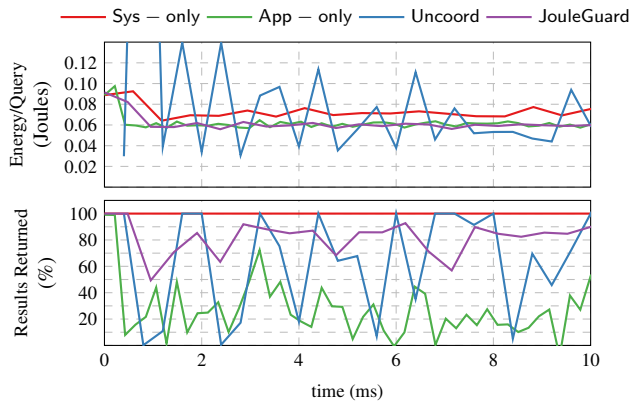

Figure 1. Different approaches to meeting and energy goal for the swish++ search engine.

that provide other guarantees and see if small changes can provide energy guarantees. In this section, we first modify an existing system-level approach, then an application-level approach. We next deploy an application and system-level approach simultaneously. All are found unsatisfactory for various reasons, so the final section previews JouleGuard. The behavior of all four approaches is illustrated in Fig. 1.

2.1 System-level Approach

The system can change swish++'s resource allocation to reduce energy consumption and a number of system-level approaches have been proposed [22, 23, 27, 43, 51, 57, 65, 68]. Among these, Cinder [51] comes closest to meeting our needs as it provides energy guarantees for individual system components. It is a user's responsibility, however, to request energy from individual components such that the total system energy respects the guarantee. To provide an overall system energy guarantee users must know precisely how much energy to request from each component and how much performance they will get from that combination. To determine the best overall configuration on our system, we must search the entire configuration space to determine if there is a combination of cores, clockspeed, hyperthreading, and memory controllers that meets the energy goal. (For more detail on the difficulty of finding the most energy efficient configuration for an application on this system, see Sec. 4.) Using the configuration found by brute force, swish++ processes 1750 qps at a cost of 125 Watts, or 0.07 Joules per query. This value is 20% higher than the goal, although this system-level approach results in no accuracy loss (as shown in Fig. 1).

The system-level approach has two drawbacks. First and foremost, it did not meet the goal. By itself, the system simply cannot reduce energy consumption to 0.06 Joules per query. Second, obtaining any energy reduction requires a tremendous knowledge of the system to request the best combination of different components. In this example, we exhaustively searched the space. In practice, we will need a more intelligent way to determine the system configuration.

2.2 Application-level Approach

Finding the system-level approach insufficient for our needs, we turn to approximate applications. Several application-level frameworks trade accuracy for other benefits [1, 5, 25, 53, 58]. None provide formal energy guarantees, but PowerDial guarantees performance [25]. We can use this performance guarantee, plus the knowledge of the system's default power consumption to meet our energy goal. We tell PowerDial to operate at 4700 qps knowing the default power is 280 Watts. Doing so, we obtain 0.06 Joules per query – exactly on target – but at a high cost of accuracy loss. On average, each query returns 83% fewer results.

2.3 Uncoordinated Application and System

The application-level approach met the energy goal, but with high accuracy loss. The system-level approach shows there are more energy-efficient system configurations. It is tempting, then to combine these approaches and meet the energy goal by decreasing system power and increasing application performance. A straightforward approach uses the application and system-level solutions concurrently, but without any communication between the two.

The problem with this uncoordinated approach is that both the application and system act without knowledge of each other. Prior work demonstrates that uncoordinated deployment of adaptive systems leads to instability and oscillatory behavior, *even when the individual adaptive components are provably well-behaved* [18, 20]. Indeed, this oscillatory behavior is demonstrated for the uncoordinated approach in Fig. 1. This oscillatory behavior results in an average performance of 2080 qps, an average power of 147 Watts, and an average return of 81% fewer results than the default. Rather than improving over application or system alone, the uncoordinated combination achieves the same energy efficiency as the system only approach with an accuracy loss close to the application-only approach.

The intuition behind the oscillatory behavior is the following. Both the application and system reason about performance *under the assumption that no other actor is manipulating performance.* When application and system act concurrently without knowledge of each other, this assumption is violated and instability occurs.

2.4 Coordinated Application and System Approaches

Rather than abandon the combination of application and system approaches, we propose actively coordinating the two. JouleGuard's coordinated approach is detailed in the next section, but we show the results here for swish++. JouleGuard uses machine learning to find the most energy efficient system configuration, which provides the performance of 1750 qps at 125 Watts. Recall, this configuration resulted in 0.07 Joules per query, 20% above the target. Therefore,

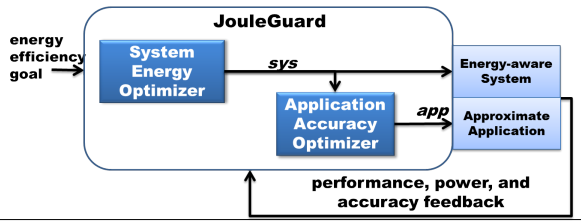

Figure 2. The JouleGuard runtime system.

JouleGuard then uses control theory to speedup the application by an additional 20% to 2100 qps. Thus, JouleGuard hits the target of $0.06 = 125/2100$ Joules per query – at a cost 24% fewer results per query than the default setup of application and system.

Of all four approaches we have explored, this coordinated one is clearly the best. It meets the energy goal while delivering the smallest possible loss in accuracy – a significant savings over the application-level approach.

2.5 Lessons Learned

This example highlights the key intuition into optimizing accuracy within a guaranteed energy budget:

- The best solution 1) finds the most energy efficient system configuration and then 2) sacrifices accuracy for any additional speedup required to meet the energy goal.
- The application and system must be coordinated to avoid unpredictable, deleterious effects.

The next section formally addresses these issues.

3. JouleGuard

This section presents JouleGuard, illustrated in Fig. 2. Following the example of the previous section, we split the problem of meeting an energy goal while maximizing accuracy into two components. The first, labeled *System Energy Optimizer* (SEO) in the figure, is responsible for putting the system into the most energy efficient system configuration. The expected speed and power of this configuration are passed to the *Application Accuracy Optimizer* (AAO), which determines how much additional performance must come from tuning application accuracy.

If the performance, power, and accuracy of all combinations of application and system configuration are known ahead of time and do not change, then the application and system configuration need only be configured once. In general, however, we expect unpredictable dynamic fluctuations making it impossible to predict the highest energy efficiency system configuration ahead of time. Furthermore, this configuration may be both application and input dependent [19, 34, 43, 44]. Therefore, we solve the optimization at runtime using dynamic feedback. Both the SEO and AAO adapt to changes in the other, yet still converge to reliable steady-state behavior. This section first describes SEO and AAO, then formally analyzes JouleGuard's guarantees.

	Symbol	Meaning
General	Sys	set of all system configs
	App	set of all application configs
	sys	an individual system config
	app	an individual application config
	$bestsys$	most energy efficient system config
	$bestapp$	most accurate app config achieving required speedup
	$default$	the default config of application or system
	r	computation rate
	p	power consumption
	a	accuracy
	s	speedup
	f	factor by which to decrease energy consumption
	W	application workload
	E	energy budget
	v_{sys}	represents variable v in system config sys
	v_{app}	represents variable v in application config app
	$\bar{v}$	represents a measured value of variable v
	$\hat{v}$	this symbol represents an estimated value of variable v
	$v(t)$	this denotes the value of v at time t
Learning	α	parameters balancing exploration and exploitation
	x	
	ρ	
	ϵ	
Control	$pole$	pole of control system
	$error$	error between required speed and current speed
	δ	multiplicative error in learned models
	$C(z)$	Z-transform of the controller
	$A(z)$	Z-transform of the application
	z	unit delay operator

Table 1. Notation used in the paper.

3.1 Notation

Table 1 summarizes the notation used throughout this section. The table has three parts. One contains general notation that is necessary throughout this section. The other two contain notation specific to either the learning or control pieces. As shown in the table, we distinguish between measured, estimated and true values of variables. For variable v, $\bar{v}$, $\hat{v}$ and v represent these three characteristics, respectively. We use subscripts to refer to the value of a variable in different configurations of the application or system. We use parenthetical notation to refer to values of variables at particular times.

3.2 System Energy Optimization

JouleGuard uses reinforcement learning to identify the most energy efficient system configuration, employing a *bandit-based* approach [32]. We model system configurations as arms in a multi-armed bandit (essentially levers in different slot machines). The reward for pulling an arm is the energy efficiency of that configuration. Our goal is to quickly determine which arm (configuration) has the highest energy efficiency. Specifically, JouleGuard estimates system configuration sys's energy efficiency by estimating performance and power $\hat{r}_{sys}(t)$ and $\hat{p}_{sys}(t)$ using exponentially weighted moving averages:

$$\begin{aligned} \hat{p}_{sys}(t) &= (1-\alpha)\cdot\hat{p}_{sys}(t-1) + \alpha\cdot\bar{p}_{sys}(t) \\ \hat{r}_{sys}(t) &= (1-\alpha)\cdot\hat{r}_{sys}(t-1) + \alpha\cdot\bar{r}_{sys}(t) \end{aligned} \quad (1)$$

We use $\alpha = .85$, which provides the best outcomes on average across all applications and systems.

In a typical bandit problem, the initial estimates might be random values. This is not a good choice for estimating performance and power as we know a general trend: power and performance tend to increase with increasing resources. Therefore, JouleGuard initializes its performance and power estimates so that the performance increases linearly with increasing resources and power increases cubically with increasing clockspeed and linearly with increasing cores. This is an overestimate for all applications, but it is not a gross overestimate. Such an initialization performs exceedingly well in practice.

The final component of a bandit solution is balancing exploration (*i.e.,* trying different configurations) and exploitation (*i.e.,* making use of the best configuration found so far). In addition, JouleGuard must be reactive to changes caused by application-level adaptation. Therefore, JouleGuard explores the system configuration space using Value-Difference Based Exploration (VDBE) [62]. VDBE balances exploration and exploitation by dynamically computing a threshold, $\epsilon(t)$ where $0 \leq \epsilon(t) \leq 1$. When selecting a system configuration, JouleGuard generates a random number $rand$ ($0 \leq rand < 1$). If $rand < \epsilon(t)$, JouleGuard selects a random system configuration. Otherwise, JouleGuard selects the most energy efficient configuration found so far. ϵ is initialized to 1 and updated every time the runtime is invoked. A large difference between the measured efficiency $\bar{r}_{sys}(t)/\bar{p}_{sys}(t)$ and the estimate $\hat{r}_{sys}(t)/\hat{p}_{sys}(t)$ results in a large ϵ, while a small difference makes ϵ small. At each iteration of the runtime $\epsilon(t)$ is updated as:

$$\begin{aligned} x(t) &= e^{\frac{-\left|\alpha\left(\frac{\bar{r}_{sys}(t)}{\bar{p}_{sys}(t)} - \frac{\hat{r}_{sys}(t)}{\hat{p}_{sys}(t)}\right)\right|}{5}} \\ \rho(t) &= \frac{1-x(t)}{1+x(t)} \\ \epsilon(t) &= \frac{1}{|Sys|} \cdot \rho(t) + \left(1 - \frac{1}{|Sys|}\right) \cdot \epsilon(t-1) \end{aligned} \tag{2}$$

If the random number is below $\epsilon(t)$, JouleGuard selects a random system configuration. Otherwise, JouleGuard searches for the system configuration with the highest estimated energy efficiency:

$$bestsys = \underset{sys}{\operatorname{argmax}} \left\{ \frac{\hat{r}_{sys}(t)}{\hat{p}_{sys}(t)} \;\middle|\; sys \in Sys \right\} \tag{3}$$

JouleGuard then puts the system into this configuration and uses the expected performance and power consumption to perform application accuracy optimization.

This bandit-based approach has the nice property that when the system models are correct $\epsilon(t) = 0$, so JouleGuard will not randomly explore the space; *i.e.,* JouleGuard will not use a random configuration after it has learned accurate models. If the system is disturbed in anyway, or the application has an unexpected impact on system performance and power, the models will be inaccurate and $\epsilon(t)$ will increase, so JouleGuard will likely explore new states to find more efficient configurations. This learning mechanism makes JouleGuard extremely robust to external variations, but it is stable when the system does not vary.

3.3 Application Accuracy Optimization

Given the system configuration found above (from Eqn. 3, JouleGuard determines the application configuration that will meet the energy goal while maximizing accuracy. Given the system performance and power estimates determined by SEO ($\hat{r}_c$ and $\hat{p}_c$) and a factor f by which to decrease energy consumption, JouleGuard must find the application configuration that provides speedup:

$$s(t) = f(t) \cdot \frac{\hat{r}_{default}}{\hat{p}_{default}} \cdot \frac{\hat{p}_{bestsys}(t)}{\hat{r}_{bestsys}(t)} \tag{4}$$

The difficulty is that ensuring energy requires that JouleGuard maintains this performance despite unpredictable events (*e.g.,* application workload fluctuations), temporary disturbances (*e.g.,* page faults), or unmodeled dependences between application configuration and system power consumption. Therefore, JouleGuard continually adjusts the speedup applied as a function of time t. JouleGuard models the problem of meeting speedup s as a control problem and minimizes the error $error(t)$ between the measured performance $\bar{r}(t)$ and the required performance $r(t) = f(t) \cdot \hat{r}_{bestsys}(t)$ at time t; *i.e.,* $error(t) = r(t) - \bar{r}(t)$.

Maintaining performance despite dynamic environmental changes is a classical control problem; many cross-layer approaches incorporate control for this reason [16, 20, 36, 59, 63, 67]. JouleGuard builds on these examples, formulating a proportional integral (PI) controller eliminating $error(t)$:

$$s(t) = s(t-1) + \frac{(1 - pole(t)) \cdot error(t)}{\hat{r}_{bestsys}(t)} \tag{5}$$

Where $s(t)$ is the speedup required beyond $\hat{r}_{bestsys}$, and $pole(t)$ is the *adaptive pole* of the control system, which determines the largest inaccuracy in the system model that JouleGuard can tolerate while maintaining stability and ensuring that the energy goal is met. While many systems use control, JouleGuard's approach is unique in that the controller constantly adapts its behavior to account for potential inaccuracies introduced by the learning mechanism.

The formal mechanism that sets the pole is discussed in Sec. 3.4). Intuitively, control techniques make it possible to determine how inaccurate the models can be and still stabilize at the goal (*i.e.,* meet the target energy while avoiding the oscillations shown for the uncoordinated approach in Fig. 1) [13]. The learning system used in SEO is constantly measuring the inaccuracy between its models and the actual performance and power (see Eqn. 2). JouleGuard uses this measured difference to set the pole. When the difference is large, the controller acts slowly, avoiding oscillations and allowing SEO to learn independently of any application changes. When the system model inaccuracy is low, the pole is small and the controller works quickly.

In summary, JouleGuard determines the application configuration by 1) measuring the performance at time t, com-

puting the error between the required and measured performance, then computing a speedup $s(t)$. JouleGuard then searches application configurations on the Pareto-optimal frontier of performance and accuracy tradeoffs to select the highest accuracy configuration delivering that speedup:

$$bestapp = \underset{app}{\text{argmax}} \{a_{app} | s_{app} > s(t) \wedge app \in A\} \quad (6)$$

3.4 Control Theoretic Formal Guarantees

JouleGuard's control system provides formal guarantees of energy consumption. We first show that the the control system converges to the desired speedup. This can be done through standard analysis in the Z-domain [35].

3.4.1 Stable and Convergent

We want to analyze the behavior of the closed loop system that maps the performance goal r into measured performance $\bar{r}(t)$. In this section we perform the analysis for a fixed pole $pole$. This analysis is necessary to understand how to adaptively set the pole in the next section.

The Z-transform of the application is simply $A(z) = \frac{\hat{r}_{bestsys}}{z}$. The Z-transform for the control system's transfer function is $C(z) = \frac{(1-pole)z}{(z-1)}$. Therefore, the transfer function of the closed loop system is:

$$\begin{aligned} F(z) &= \frac{C(z) \cdot A(z)}{1+C(z) \cdot A(z)} \\ &= \frac{\frac{(1-pole)z}{(z-1)} \cdot \frac{\hat{r}_{bestsys}}{z}}{1+\frac{(1-pole)z}{(z-1)} \cdot \frac{\hat{r}_{bestsys}}{z}} \\ &= \frac{1-pole}{z-pole} \end{aligned} \quad (7)$$

Following standard control analysis, we can evaluate the system's stability and convergence to the goal [35]. The system is stable, in the control theoretic sense, that it will not oscillate around the goal if and only if $0 \leq pole < 1$. Therefore, $pole$ must be chosen to meet these restrictions. Furthermore, the system is convergent, meaning that when it stabilizes $error(t) = 0$ if and only if $F(1) = 1$. From Eqn. 7 we can see that this condition is clearly met. We therefore conclude that the control system is stable and convergent. These guarantees, however, are based on the implicit assumption that $\hat{r}_{bestsys}$ is an accurate estimate of the performance delivered in $bestsys$. In the next section we discuss how to ensure stability even when the estimate is inaccurate (as it likely is at system initialization).

3.4.2 Robust to Learning Inaccuracies

We determine the controller's robustness to inaccurate estimates of $\hat{r}_{bestsys}$ by analyzing Eqn. 7. Suppose the estimate is incorrect and the true value is $r_{sys} = \delta \hat{r}_{sys}(t)$ where δ is a multiplicative error in the estimation. For example, $\delta = 5$ indicates that model is off by a factor of 5. We determine JouleGuard's robustness to these inaccuracies by substituting $\delta \hat{r}_{sys}$ into $F(z)$:

$$\begin{aligned} F(z) &= \frac{C(z) \cdot A(z)}{1+C(z) \cdot A(z)} \\ &= \frac{\frac{(1-pole)z}{(z-1)} \cdot \frac{\delta \hat{r}_{sys}}{z}}{1+\frac{(1-pole)z}{(z-1)} \cdot \frac{\delta \hat{r}_{sys}}{z}} \\ &= \frac{(1-pole)\delta}{z+(1-pole)\delta-1} \end{aligned} \quad (8)$$

The controller represented by Eqn. 8 is stable if and only if the pole is between 0 and 1. Thus, for a stable system

$$0 < \delta < \frac{2}{1-pole}. \quad (9)$$

So, the value of $pole$ determines how robust JouleGuard is to model inaccuracies. For example, $pole = 0.1$ implies that $\hat{r}_{sys}(t)$ can be off by a factor of 2.2 and JouleGuard will still converge.

To provide convergence guarantees – and, thus, energy guarantees – JouleGuard must set the pole to provide stability and avoid the oscillations seen in the swish++ example. JouleGuard has a bound on model inaccuracies as it is constantly updating its estimates of system performance using Eqn. 1. Thus, JouleGuard computes $\delta(t)$, the multiplicative inaccuracy at time t as:

$$\delta(t) = \left| \frac{\bar{r}(t)}{\hat{r}_{sys}(t-1)} - 1 \right| \quad (10)$$

and computes the pole as:

$$pole(t) = \begin{cases} \delta(t) > 2: & 1 - 2/\delta(t) \\ \delta(t) \leq 2: & 0 \end{cases} \quad (11)$$

JouleGuard automatically adapts the pole so that controller is robust to inaccuracies in the system models. In practice, the pole is large when the learner is unsure and likely to randomly explore the space. This means that the controller will be slow to change configurations when the learner is aggressive. In contrast, when the learner converges the inaccuracy is low (by definition) and the controller can be more aggressive. *This adaptive pole placement combined with machine learning is the unique feature of JouleGuard which distinguishes it from prior approaches and allows JouleGuard to split the energy guarantee problem into two subproblems yet still provide robust guarantees.*

3.4.3 Impossible Goals

A user may request an energy goal that is impossible to meet given the application and the system. In this case, JouleGuard reports that the goal is infeasible and then configures application and system to provide the smallest possible energy consumption.

3.5 Implementation

JouleGuard's runtime is summarized in Algorithm 1. We implement this algorithm through a straightforward coding of the math described above and summarized in the algorithm listing. The two key challenges are measuring feedback and configuring the application and system. These are really interface issues rather than technical issues. JouleGuard needs

Application	Configs	Speedup	Acc. Loss (%)	Accuracy Metric
x264	560	4.26	6.2	Peak Signal to Noise Ratio (PSNR) [25]
swaptions	100	100.35	1.5	swaption price [25]
bodytrack	200	7.38	14.4	track quality [25]
swish++	6	1.52	83.4	precision and recall [25]
radar	26	19.39	5.3	signal to noise ratio [21]
canneal	3	1.93	7.1	wire length [56]
ferret	8	1.24	18.2	similarity [56]
streamcluster	7	5.52	0.55	quality of clustering [56]

Table 2. Approximate Application configurations.

Algorithm 1 The JouleGuard Runtime.

Require: W ▷ Workload provided by user
Require: E ▷ Energy budget provided by user
loop
  Measure work done $W(t)$ and energy consumed $E(t)$.
  Measure performance $\bar{r}(t)$ and power $\bar{p}(t)$ in configuration c.
  Update performance and power estimates $\hat{r}_c$ and $\hat{p}_c$ (Eqn. 1).
  Update $\epsilon(t)$ (Eqn. 2).
  Generate random number $rand$
  if $rand < \epsilon(t)$ **then**
    Select random system configuration
  else
    Select energy optimal system configuration sys (Eqn. 3).
  end if
  Compute the controller's pole (Eqns. 10 and 11)
  Compute remaining energy and work.
  Use those values to compute speedup target (Eqn. 4).
  Compute speedup control signal (Eqn. 5).
  Select the application configuration to deliver speedup (Eqn. 6).
end loop

to be supplied a function that reads performance and power. Any performance metric can be used as long as it increases with increasing performance. Similarly, power can be read from an external power monitor or from modern hardware devices that support on-board power measurement. Prior work defined a general interface for specifying system-level configurations [23]. A straightforward extension of this interface allows it to support application configuration changes as well. Given these interfaces, we implement JouleGuard as a C runtime that can be compiled directly into an application. It can replace existing runtime systems for approximate applications, or it can convert a statically configured approximate application into one dynamically configurable to meet energy goals.

3.6 Application Accuracy Requirements

JouleGuard does not require precise quantification of application accuracy, rather it requires an ordering on application configurations. Many frameworks provide precise accuracy quantification (*e.g.,* [5, 24, 25, 48]), others do not (*e.g.,* [9, 14, 58]), and some leave it to the programmer (*e.g.,* [38, 53]). Approaches that do not quantify accuracy still order configurations, but the order represents preferences rather than absolute numerical differences. JouleGuard never needs a precise value for accuracy. The only place it reasons about accuracy is in Eqn. 6 when selecting an application configuration. This equation only requires a total order on available configurations. Thus, JouleGuard is compatible with a wide range of approximate approaches including those that do not specifically quantify accuracy.

3.7 Modification for Approximate Hardware

This section has assumed that all accuracy tradeoffs occur at the application level. Recent proposals, however, propose approximate hardware that provides reduced energy consumption in exchange for occasionally returning the wrong result [4, 11, 12, 38, 45, 46]. In most cases, these approximate hardware implementations maintain the same timing, but reduce power consumption. For those cases, it is straightforward to modify the above control system to work with approximate hardware. We would begin by using the learning engine to find the most energy efficient system configuration that sacrifices no accuracy (this step is the same as the above). We then modify the JouleGuard control system to manage power (rather than speedup) by tuning hardware level approximation. The approach would be very similar, but the controller would reduce power instead of increase performance. Yet another problem would be to coordinate approximate applications with approximate hardware. Such a combination likely requires a significant modification of JouleGuard.

4. Experimental Setup

To demonstrate generality, we test JouleGuard with eight different applications on three different hardware platforms.

4.1 Applications

We draw on existing approximate applications from two sources. The first is PowerDial, which automatically turns static command line parameters into a data structure which alters runtime performance and accuracy tradeoffs [25]. The second is Loop Perforation, which eliminates some loop iterations to trade accuracy for performance [56]. We build x264, swaptions, bodytrack, radar, and swish++ with PowerDial. We build canneal, ferret, and streamcluster with Loop Perforation. Table 2 summarizes the available application configurations, showing the total available configurations, the maximum speedup, maximum accuracy loss (as a percentage of the default value), and the accuracy metric.

System	Configuration	Settings	Speedup	Powerup
Mobile	clock speed	8	2.72	1.94
	big cores	4	4.52	2.00
	big core speeds	19	10.23	10.42
	LITTLE cores	4	4.52	1.32
	LITTLE core speeds	13	7.11	2.62
Tablet	clock speed	8	2.72	1.94
	core usage	2	1.81	1.22
	hyperthreading	2	1.10	1.03
	idle	n/a	1.00	1.00
Server	clock speed	16	3.23	2.05
	core usage	16	15.99	2.03
	hyperthreading	2	1.92	1.11
	idle	n/a	1.00	1.00
	mem controllers	2	1.84	1.11

Table 3. System configurations.

These applications represent several different workloads. x264 is a video encoder, which can trade increased noise in the output video for increased frame rate. swaptions is a financial analysis benchmark that trades accuracy of price for speed of pricing. bodytrack does image analysis to follow a person moving through a scene. It trades the precision of the track for increased throughput. The swish++ search engine is a webserver which supports document search and can trade the precision and recall of the search results for decreased search time. radar is a digital signal processing program that detects targets in the returns of a phased array antenna [21]. canneal is an engineering application that performs place-and-route on a netlist; it can trade increased wire length for decreased routing time. ferret is a image similarity search that can decrease the similarity of the results it returns in exchange for decreased search time. Finally, streamcluster is a clustering algorithm that can decrease the quality of its clustering for increased performance. Each application supports a different accuracy metric. To standardize the presentation, we report accuracy as a proportion of that achieved when running in the application's default configuration; *i.e.,* without PowerDial or Loop Perforation.

These benchmarks might easily be run in an environment where predictable energy consumption is essential for successful operation. For example, x264, bodytrack, and radar might be executed on either a mobile or embedded device with limited battery; ferret and swish++ might be executed on a server where administrators want to deliver better results to higher paying users. We note that swish and canneal do not run on Mobile, but it is unlikely that a user would want to run a webserver or engineering application on their mobile phone.

4.2 System

In the introduction, we argue that maximizing accuracy on an energy budget could be useful for both mobile and server systems. Therefore, we evaluate JouleGuard on three platforms representative of both domains. The first, Mobile, is an ODROID-XU3 from HardKernel with a Samsung Exynos 5 heterogeneous ARM big.LITTLE processor consisting of 4 Cortex-A15 (big) cores and 4 Cortex-A7 (LITTLE) cores. It has 2GB of RAM and a 64 GB SD card for permanent storage. The second, Tablet, is a Sony Vaio SVT11226CXB tablet with an Intel i5-4210Y processor, 4GB RAM, and a 128GB SSD. The third, Server is a dual socket system with two eight-core Intel Xeon E5-2690 processors, 64GB RAM, and a 256 GB HDD. We monitor power and energy on Mobile with INA-231 power sensors that provide power data for the big Cortex-A15 cluster, the LITTLE Cortex-A7 cluster, the DRAM and the GPU. Both Intel platforms support hyperthreading and TurboBoost, and both allow power and energy consumption to be read directly from registers at runtime in millisecond intervals [50]. Mobile idles at approximately .12 Watts, Tablet idles at 2.4 Watts, and Server approximately 12 Watts. The maximum power consumption on the mobile processor is 6 Watts, on the tablet 9 Watts, and 270 Watts on the server. For the Intel systems, in addition to the on-chip power meters, we use an external power meter to measure the power consumption of the other components in each system. The mobile system has an additional 5.8 Watts of power consumption beyond the processor; the server system has an additional 75-90 Watts.

To provide energy guarantees, we must account for the full energy consumption of the system. This presents a problem for both the Intel-based systems as their total energy consumption is much higher than what the on-chip monitors report. We account for this by using the on-chip monitors (which return results at a millisecond granularity) and simply adding a fixed amount of power to this quantity to account for the additional power consumption. The external power meters are too slow to provide dynamic feedback (1s granularity), but they can be used to verify total system power and energy. Therefore, we use the on-chip power meters plus a fixed constant for dynamic feedback and use the external power meters to measure the full system energy over the life of the experiments. All measurements in the evaluation section report full system power and energy.

Mobile runs Ubuntu Linux 14.04 using a modified kernel 3.10.58+. The other systems run Linux 3.2.0 with the `cpufrequtils` package to change clock speeds. We use process affinity masks to assign an application to cores and hyperthreads. We use the `numalib` package to assign memory controllers to an application on the server. The available system configurations on all three platforms are summarized in Table 3, which shows the total number of available configurations and the maximum increase in speed and power measured on each machine. There are effectively an unlimited number of idle settings, as any application could be stalled arbitrarily.

4.3 System Characterization

The JouleGuard framework contains a learning engine to determine the most energy efficient system configuration.

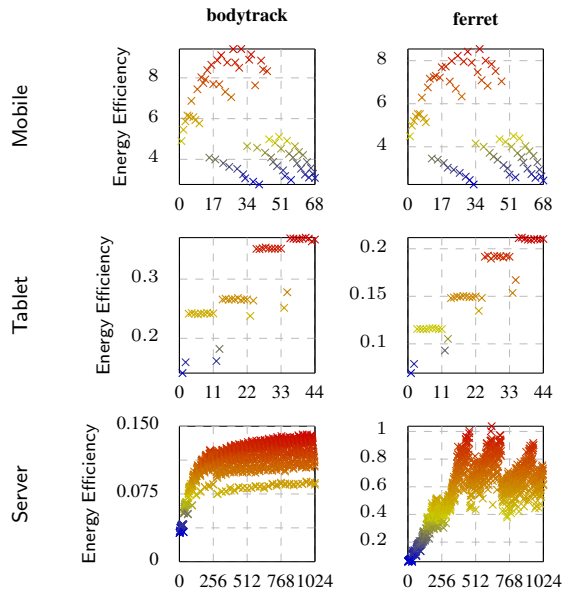

Figure 3. Example energy efficiencies.

In this section, we justify the need for this approach by characterizing our platforms. We run each application in its default (full accuracy) mode and measure the energy efficiency (performance divided by power consumption) for each possible system configuration.

The results for two example applications (bodytrack and ferret) are shown in Fig. 3. All three systems contain multiple configurable resources, so we linearize the configuration space into a configuration index, which is shown on the x-axis of each chart. The y-axes show the energy efficiency. We have chosen configuration indices so that the highest index always represents all resources assigned to an application at their highest setting (this is the default configuration for both systems). The lowest index is always a single core at the slowest clockspeed. The configuration indices for different applications on the same platform are the same; *i.e.,* index i for bodytrack represents the same resource usage as index i for ferret.

The two applications are representative of the extremes in our benchmark set, the others are omitted to save space. bodytrack is generally easy to manage as it presents smooth curves with no local extrema. ferret is harder to manage and exhibits complicated behavior on the most configurable system, Server.

Examining this data, it is significant to compare the peak energy efficiency to the default system configuration (the highest index) for a given application. It is also important to compare the location of the peak across different applications. We observe:

- All applications show significant differences between the highest and lowest energy efficiency on all platforms. Thus, there is a penalty for choosing the system configuration incorrectly.
- On Mobile, the large cores are the least energy efficient (they are clustered in the bottom right of the chart. On this system, the learner will need to move off the big cores and recognize the importance of using smaller cores for energy savings.
- On the Tablet platform, the peak energy efficiency occurs at the default setting for all applications. This implies it should be easy to select the peak for that platform. In addition, many of the clockspeed settings appear to produce the same energy efficiency. It appears that the firmware on this platform disables most of the clockspeeds that the processor supports. The real difficulty of managing this system will be for JouleGuard to recognize that many settings produce identical results.
- On the Server platform, each application has a unique configuration providing maximum energy efficiency, implying it is much harder to select the most energy efficient configuration on that platform. Furthermore, none of our test applications achieve peak energy efficiency in the default configuration; *i.e.,* the default is wasteful.

These results broadly indicate that it should be easy to perform the system energy optimization on Mobile and Tablet and difficult on Server. The difference in platform characteristics creates a unique challenge: the learner should be low-overhead so that it does not greatly impact the behavior of the easy platform and agile so that it can overcome the difficulties imposed by the difficult platform. The next section evaluates JouleGuard' ability to overcome the challenges of these platforms.

5. Evaluation

This section evaluates JouleGuard. We begin by measuring the runtime overhead. We then demonstrate that JouleGuard converges to the desired energy consumption and does so with near optimal accuracy. Next, we compare JouleGuard's coordinated approach to the best accuracy that can be achieved by application or system alone. Finally, we show that JouleGuard maintains energy goals during application phase changes.

5.1 JouleGuard Overhead

We expect JouleGuard to be low-overhead as the algorithm detailed in Algorithm 1 is low complexity. The most expensive parts involve searching for particular configurations according to Eqns. 3 and 6, which we implement with binary search. To measure overhead, we time 100 iterations of the runtime managing x264 (the application with the largest number of application-level configurations) on each of the three platforms. The time per iteration (in microseconds) is shown in

Platform	Latency (μs)
Mobile	249
Tablet	164
Server	82

Table 4. Runtime overhead.

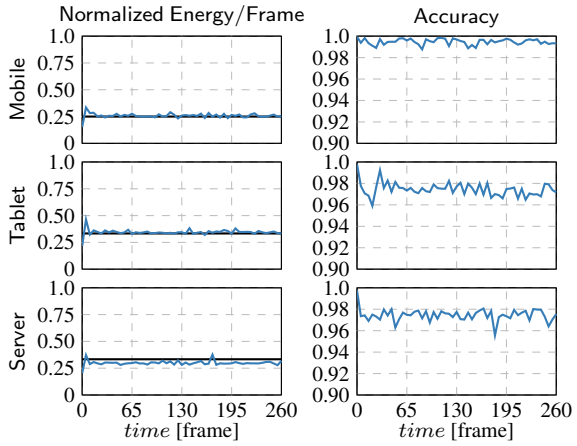

Figure 4. JouleGuard stabilizes to a consistent energy consumption subject to application and system noise.

Table 4. The mobile system has less work to do (because it has the fewest configurations to consider), but it also has the smaller computational capacity. Regardless, these overheads are small enough to permit millions of runtime iterations per second. This is far faster than we can receive power feedback, so JouleGuard's overhead is negligible for its intended use.

5.2 Methodology

For each application and platform, we first measure accuracy and energy consumption in the default configuration; *i.e.,* we run out-of-the-box on our platforms with no changes. Having established a baseline energy consumption, we then deploy each application with JouleGuard for a several energy goals which decrease energy by some factor f where $f \in \{1.1, 1.2, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0\}$. For example, $f = 2.0$ reduces energy consumption by $2\times$ compared to the default. For each goal we measure both the achieved energy and accuracy.

To quantify JouleGuard's ability to meet energy goals, we compute *relative error*:

$$Relative\ Error = \begin{cases} \bar{e} > e_{goal}: & \frac{\bar{e} - e_{goal}}{e_{goal}} \cdot 100\% \\ \text{otherwise}: & 0 \end{cases} \quad (12)$$

between the measured energy and the goal. We only count the error if it is above the target; *i.e.,* JouleGuard consumed too much energy. Energies below the target, however, will likely result in less than optimal accuracy. Low relative error indicates the energy consumption is close to the target, while high error means JouleGuard missed the target substantially. Relative error is a percentage, allowing comparisons across different targets.

To quantify optimality, we construct an *oracle* representing the best possible accuracy for an application, system, and energy target. We exhaustively profile the application and system in every possible configuration to determine the optimal accuracy for different energy targets. We then calculate the best system and application state for every application iteration. The oracle thus represents the best accuracy that could be accomplished by dynamically managing application and system with perfect knowledge of the future and no overhead. We quantify optimality as *effective accuracy:*

$$effective\ acc = \bar{a}/a_{oracle}(goal) \quad (13)$$

Where $\bar{a}$ is the measured accuracy for the application running with JouleGuard and $a_{oracle}(goal)$ represents the accuracy our oracle returns for the given energy goal.

5.3 Stability and Convergence

One of JouleGuard's essential properties is convergence to desired energy targets. Previous sections demonstrate convergence analytically, this section evaluates it empirically.

We begin by showing a representative application operating under JouleGuard. Fig. 4 shows bodytrack's energy per frame (in the left column) and achieved accuracy (in the right column). Energy per frame measures how well JouleGuard tracks the goal. In these figures, JouleGuard is maintaining an average energy per frame of $1/4$ the default for Mobile and $1/3$ the default for the other systems (*i.e.,* $f = 3$, representing a three fold decrease in energy consumption). The solid black lines represent the target energy per frame. These figures clearly demonstrate that JouleGuard can track the desired energy. They also show how accuracy is affected by the target platform – Tablet and Server have less energy efficient configurations available so they must sacrifice more accuracy to meet a less aggressive goal.

To demonstrate the stability and convergence in general we compute relative error for all applications, hardware platforms, and energy targets. These results are shown in Fig. 5 with benchmark name on the x-axis and relative error (as a percentage) on the y-axis. There is a bar for each energy target if the target is feasible (*e.g.,* given the available application configurations ferret can only achieve reductions up to $1.2\times$ on Tablet and Server). In general, relative error is low – demonstrating that JouleGuard meets energy guarantees on a number of platforms for a number of applications. Generally, the more aggressive the target the higher the error. This relationship is not surprising as, if the controller makes an error in one iteration it may be hard to make up for it in another iteration with highly aggressive energy targets.

5.4 Optimality

We now quantify accuracy for all benchmarks, platforms, and energy targets by presenting the effective accuracy calculated according to Eqn. 13.

These results are shown in Fig. 6. The effective accuracy is close to unity, representing accuracy very close to the oracle. This accuracy is achieved despite JouleGuard's overhead and the inherent noise in the benchmarks. The only benchmark that causes some issues is swish++ on Tablet at the $1.5\times$ energy efficiency mark. In this case, JouleGuard meets

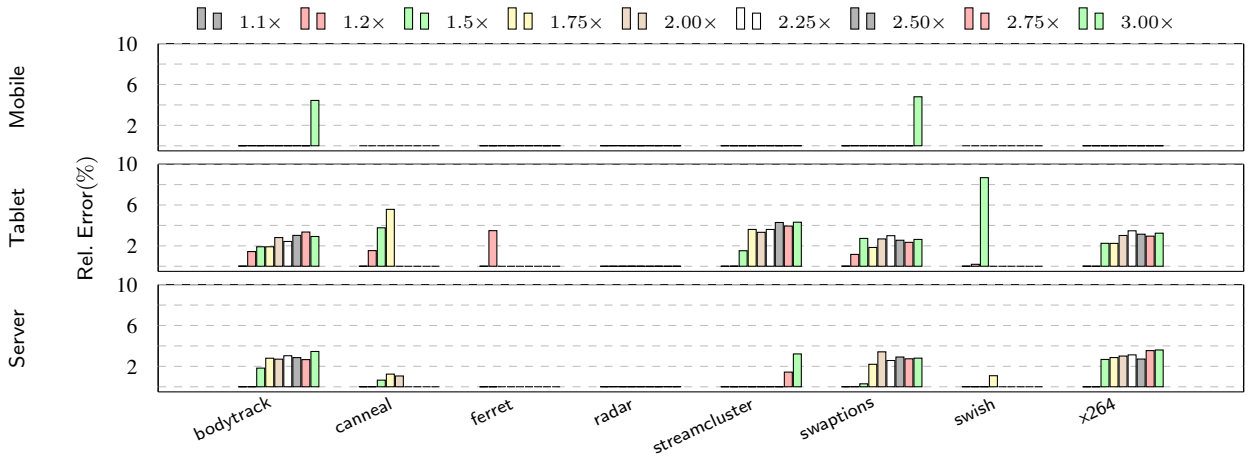

Figure 5. JouleGuard's low relative error shows it is within a few % of the desired energy.

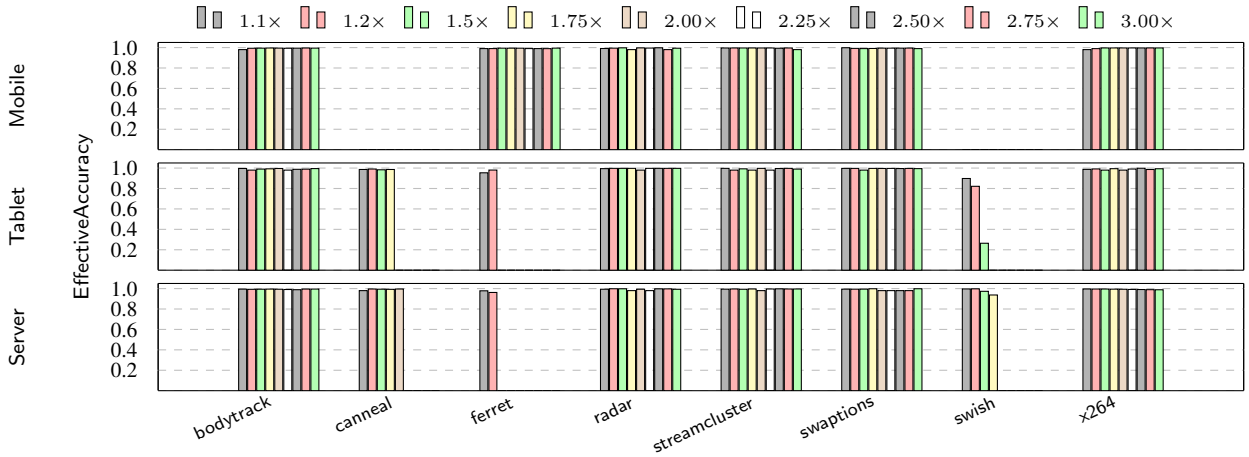

Figure 6. JouleGuard achieves near optimal accuracy for different energy goals.

the energy requirement (see Fig. 5), but sacrifices more accuracy than strictly necessary. This lack of optimality is related to a point made above: this energy target represents the extreme operating range for this application and platform. While our oracle never makes mistakes, JouleGuard does. In this case, missing an energy goal in one iteration requires JouleGuard to make it up in others. As JouleGuard is already operating near the limit, making up energy requires further sacrifices of accuracy. Furthermore, the accuracies for Mobile are uniformly higher showing almost no deviation from the oracle. This level of optimality on that platform is due to the fact that all the energy goals are well within the operating range for that platform (see Fig. 3 for examples of the much greater range of operating points on Mobile).

5.5 Comparison to Other Approaches

Many existing frameworks for approximate computing consider only the application [5, 25, 56, 58]. Similarly, many energy optimization approaches consider only system resource usage [27, 43, 51, 57]. We now compare the accuracy achieved by JouleGuard's coordinated approach to the best possible accuracy that could be achieved by adapting just application or system alone. Recall Sec. 2 describes how to maximize accuracy for an application or energy efficiency for a system in isolation. In this section, we run JouleGuard for a number of energy targets and compare its accuracy to the best possible that considers only application-level adaptation. In addition, we compare the best possible energy savings that can be achieved by manipulating the system alone.

Fig. 7 shows the results for Server. Results for the other systems are omitted for space. Each chart shows the energy savings goal on the x-axis and the achieved accuracy for both JouleGuard and the application only approach. In addition, a dotted horizontal line shows the maximum range of increased energy savings possible by manipulating the system's resource usage only. In most cases, the system-only approach reduces performance for decreased energy. In some cases, (*e.g.,* ferret – see Fig. 3), the system can find

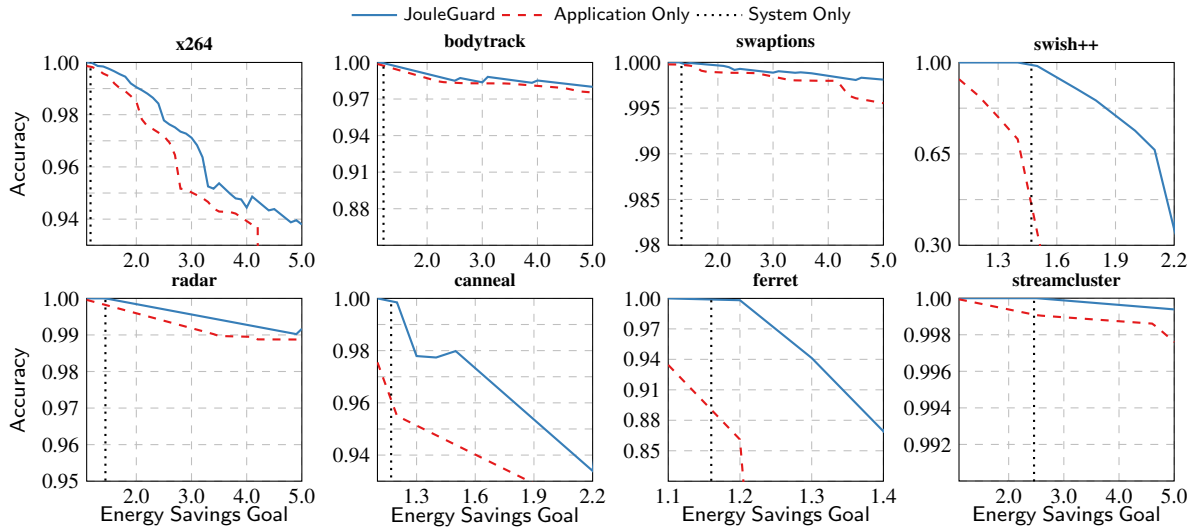

Figure 7. Comparison of JouleGuard and application- or system-only approaches on Server (higher is better).

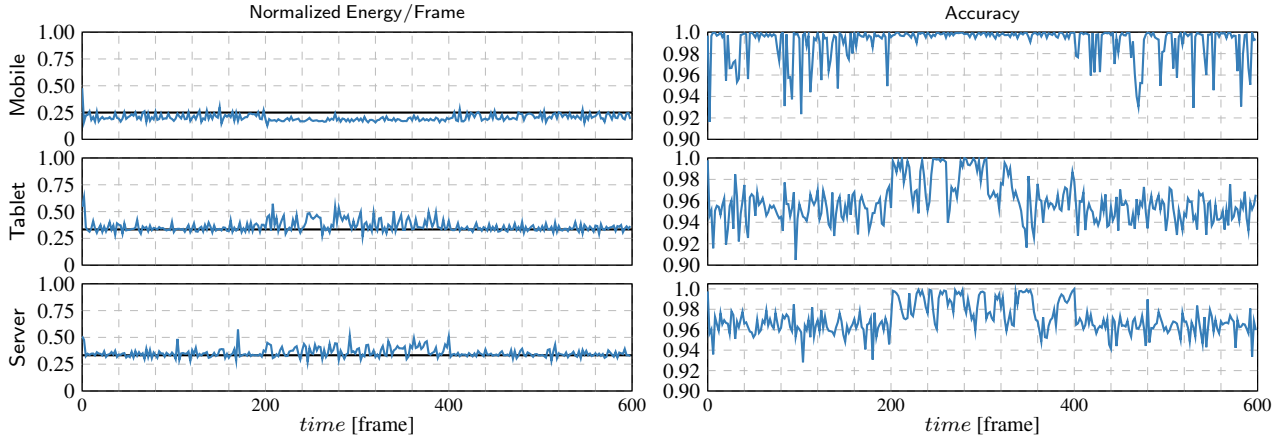

Figure 8. JouleGuard adapts to application phases.

a more energy efficient configuration that is faster than the default. If no data is shown for an energy goal, it is not feasible. These results show that proactive coordination – used by JouleGuard – provides uniformly higher accuracy for the same energy compared to approaches that work at application-level only. In addition, JouleGuard's combination of application and system-level optimization provides a greater range of possible energy efficiencies than can be achieved by either alone. In addition, these results show that JouleGuard does not needlessly waste accuracy – for JouleGuard, accuracy only starts to decrease at the point where system-level manipulations are no longer effective.

5.6 Reaction to Application Phases

Our final experiment shows JouleGuard adapting to application phases. In this example, we concatenate three videos together to form a new input with three distinct phases. Each phase lasts for 200 frames. The first and third phases are the same. The second phase is an easier scene that naturally (without any control) encodes about 40% faster than the first scene. Thus, the second scene naturally requires less energy than the first or third. Ideally, when it encounters the second scene, JouleGuard would maintain the same target energy per frame and turn the energy savings into increased accuracy. Doing so would clearly demonstrate the ability to maximize accuracy while ensuring energy consumption.

Fig. 8 shows the results with time (in frames) measured on the x-axes and energy per frame (normalized to the goal) on the y-axes. At frame 200, the scene changes and we see a very short spike in energy followed by an increase in accuracy as JouleGuard turns the energy efficiency gain into improved accuracy. This effect can be seen in the accuracy charts as all three platforms produce higher accuracy during the middle 200 frames.

5.7 Results Summary

These results empirically demonstrate the claims made in the introduction. Specifically, JouleGuard is:

- **Low Overhead:** it configures rapidly (Table 4).
- **Convergent:** it meets a range of energy goals with low error for all applications (built from two different approximate computing frameworks) on two different hardware platforms (Fig. 5).
- **Near-Optimal:** it achieves close to the best possible accuracy for almost all applications across the different platforms and energy goals (Fig. 6). In addition, JouleGuard's proactive approach coordinating application and system is uniformly better than approaches that consider application only (Fig. 7).
- **Responsive:** it maintains energy goals with high accuracy during application phase changes (Fig. 8).

6. Related Work

We describe several related efforts in approximate applications, energy-aware systems, cross-layer approaches, and adaptive control.

6.1 Approximate Applications

Approximate applications trade accuracy for performance, power, energy, or other benefits. Approaches include both static analysis [2, 7, 8, 49, 53, 56] and dynamic support for tradeoff management [1, 5, 9, 25, 30, 31, 48, 58]. Static analysis guarantees that accuracy bounds are never violated, but it is conservative and may miss chances for additional savings through dynamic customization.

Dynamic approximation tailors runtime behavior to specific inputs. For example, Green maintains accuracy goals while minimizing energy [5], and Eon extends battery life in exchange for accuracy [58]. Both Green and Eon use heuristic techniques for managing the tradeoff space. PowerDial [25], uses control theoretic techniques to provide performance guarantees while maximizing accuracy. Each of these approaches supports a single constraint. For example, Green uses heuristics to ensure accuracy bounds are not violated, but it cannot guarantee energy consumption. PowerDial formally guarantees performance (so it can meet real-time or quality-of-service goals), but it does not manage energy. Eon uses heuristics to prevent embedded devices from running out of energy, making it well-suited for systems that harvest energy, but it does not maximize accuracy on an energy budget. Furthermore, these approaches are designed to work at the application-level only. Fig. 7 demonstrates the benefits of JouleGuard's approach which combines application and system adaptation.

6.2 Energy-aware Systems

Many system-level approaches coordinate the use of multiple resources to provide performance guarantees with reduced power or energy consumption. For example, Li et al. manage memory and processor speed [37], Dubach et al. coordinate several microarchitectural features [10], and Maggio et al. coordinate core allocation and clock speed [41]. Meisner et al. propose coordinating CPU power states, memories, and disks to meet performance goals while minimizing power consumption [42]. Bitirgen et al. coordinate clockspeed, cache, and memory bandwidth in a multicore [6]. The METE system controls cache, processor speed, and memory bandwidth to meet performance requirements [54]. Still other approaches manage arbitrary sets of system-level components [23, 47, 59, 67]; however, none of these approaches coordinates system resource usage with application-level adaptation. Furthermore, these approaches all ensure performance is met while minimizing energy consumption – none can ensuring energy budgets are met.

Several OS designs support the measurement, allocation, and management of energy as a first class object. The Koala system uses a predictive model to minimize application energy while maintaining performance [57]. Similarly, *power containers* support fine-grain tailoring of heterogeneous resources to varying workloads [55]. Cinder creates abstractions which allow energy to be stored and allocated on mobile devices [51]. These OS-level approaches all provide mechanisms for allocating energy, but not policies for performing the allocation or enforcing energy budgets, other than halting if the budget is exceeded.

6.3 Cross-layer Approaches

Static approaches coordinate application and system by marking application regions as candidates for accuracy loss and then statically determining when the system can turn that loss into performance or energy savings. The Truffle architecture [11] supports applications which explicitly mark some computations and data as "approximate." Parrot replaces approximate regions of an application with a neural network implementation, which is then executed on a special hardware neural processing unit [12]. Flikker, allows applications to mark some data as "non-critical," and store it in a DRAM that trades accuracy for energy savings [38]. The codesign of application with *inexact* hardware has been proposed to reduce energy consumption dramatically in exchange for reduced application accuracy [4, 45, 46].

JouleGuard is inspired by prior work that dynamically coordinates across application and system layers. Flinn and Satyanarayanan coordinate operating systems and applications to meet user defined energy goals [14, 15]. This system trades application quality for reduced energy consumption. The GRACE OS employs hierarchy to provide predictable performance for multimedia applications, making system-level adaptations first and then application-level adaptations [63, 66]. Like GRACE, Agilos uses hierarchy, combined with fuzzy control, to coordinate multimedia applications and systems to meet a performance goal [36]. Maggio et al. propose a game-theoretic approach for a decentralized coordination of application and system adapta-

tion which provides real-time guarantees [39]. xTune uses static analysis to build a model of application and system interaction and then refines that model with dynamic feedback [33]. The CoAdapt system uses a control theoretic approach to meet a performance, power, or accuracy constraint [20].

Two key features distinguish JouleGuard from prior cross-layer approaches. First, prior work does not provide energy guarantees, most, instead, guarantees performance while minimizing energy consumption. Second, prior work splits the system and application into two linear problems, which is possible because of the focus on performance [20, 36, 63, 66]. JouleGuard also splits the problem into two subproblems, but acknowledges that these problems are not independent. A key contribution of JouleGuard is to formulate a solution that is provably robust despite the dependence between the subproblems. JouleGuard's approach requires novel solutions for both subproblems rather than repurposing existing work to target energy.

In summary, JouleGuard complements prior work on approximate computing by adding the capability to meet energy budgets with near maximum accuracy, benefiting mobile users who need to finish a task given the current charge on their battery and desktop or server users who want to maximize the quality of a result given an energy budget.

6.4 Adaptive Control Systems

Integrating control systems into software is one way to create self-adaptive software systems [52]. Control theory provides a general technique that can be applied to tune various aspects of the software system to meet goals [17]. A classical control system provides numerous formal guarantees about its behavior; however, these guarantees are all dependent on how well the difference model used to construct the controller captures the underlying system dynamics. *Adaptive* control is an approach that allows the controller itself to be adjusted online [3].

In an adaptive control system, the controller becomes a meta-model which is dynamically tuned as the control system runs. Adaptive control has been used to tune resource usage in webservers [28, 29] and to minimize energy in embedded systems [27, 41]. Filieri et al. propose a general methodology for constructing adaptive controllers [13] and some middleware frameworks incorporate adaptive control [67]. A recent survey finds that adaptive control significantly increases generality compared to classical control [40].

Typically, adaptive control systems make the control system more robust to external variations that might invalidate a static model. In this paper, we use adaptive control to make the control system robust to another adaptive system: the learning engine. The interface between the two is the error in the learned models. As adaptive systems proliferate, it is increasingly likely that multiple adaptive systems, each designed by different engineers will be deployed concurrently. Simple interfaces which coordinate adaptive computing systems will be necessary to avoid the type of bad behavior illustrated in Sec. 2 and documented in other studies [18, 20]. This paper contributes one such interface.

7. Conclusion

This paper introduces JouleGuard, an optimizing runtime system that coordinates approximate applications and energy-aware systems to meet energy goals while maximizing accuracy. JouleGuard is based on the key insight that we can solve this particular optimization by dividing the problem into two sub-problems, each of which can be solved efficiently. The first sub-problem moves the system to the most energy efficient configuration, while the second dynamically manages performance. JouleGuard proposes a machine learning approach to the first problem and a control theoretic solution to the second. We have implemented JouleGuard and tested it with a number of applications and two systems. We find that – both empirically and analytically – JouleGuard meets energy budgets with near optimal accuracy, while adapting to application phases and consistently outperforming approaches that consider only application or system configurations.

Acknowledgments

We are grateful to Shan Lu and Anne Rogers, who both read early drafts of this work and provided extremely valuable feedback. We also thank the anonymous reviewers, who hopefully find the final version of this paper improved from the submitted version. Finally, we thank Bryan Ford for shepherding the paper.

Henry Hoffmann's effort on this project is funded by the U.S. Government under the DARPA PERFECT program, by the Dept. of Energy under DOE DE-AC02-06CH11357, and by the NSF under CCF 1439156.

References

[1] J. Ansel, M. Pacula, Y. L. Wong, C. Chan, M. Olszewski, U.-M. O'Reilly, and S. Amarasinghe. "Siblingrivalry: online autotuning through local competitions". In: *CASES*. 2012.

[2] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe. "Language and compiler support for auto-tuning variable-accuracy algorithms". In: *CGO*. 2011.

[3] K. J. Astrom and B. Wittenmark. *Adaptive Control*. 2nd. 1994. ISBN: 0201558661.

[4] L. Avinash, C. C. Enz, K. V. Palem, and C. Piguet. "Designing Energy-Efficient Arithmetic Operators Using Inexact Computing". In: *J. Low Power Electronics* 9.1 (2013).

[5] W. Baek and T. Chilimbi. "Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation". In: *PLDI*. June 2010.

[6] R. Bitirgen, E. Ipek, and J. F. Martinez. "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach". In: *MICRO*. 2008.

[7] M. Carbin, D. Kim, S. Misailovic, and M. C. Rinard. "Proving acceptability properties of relaxed nondeterministic approximate programs". In: *PLDI*. 2012.

[8] M. Carbin, S. Misailovic, and M. C. Rinard. "Verifying quantitative reliability for programs that execute on unreliable hardware". In: *OOPSLA*. 2013.

[9] F. Chang and V. Karamcheti. "Automatic Configuration and Run-time Adaptation of Distributed Applications". In: *HPDC*. 2000.

[10] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O'Boyle. "A Predictive Model for Dynamic Microarchitectural Adaptivity Control". In: *MICRO*. 2010.

[11] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. "Architecture support for disciplined approximate programming". In: *ASPLOS*. 2012.

[12] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. "Neural Acceleration for General-Purpose Approximate Programs". In: *MICRO*. 2012.

[13] A. Filieri, H. Hoffmann, and M. Maggio. "Automated design of self-adaptive software with control-theoretical formal guarantees". In: *ICSE*. 2014.

[14] J. Flinn and M. Satyanarayanan. "Managing battery lifetime with energy-aware adaptation". In: *ACM Trans. Comp. Syst.* 22.2 (May 2004).

[15] J. Flinn and M. Satyanarayanan. "Energy-aware adaptation for mobile applications". In: *SOSP*. 1999.

[16] A. Goel, D. Steere, C. Pu, and J. Walpole. "SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit". In: *2nd USENIX Windows NT Symposium*. 1998.

[17] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004. ISBN: 047126637X.

[18] J. Heo and T. F. Abdelzaher. "AdaptGuard: guarding adaptive systems from instability". In: *ICAC*. 2009.

[19] H. Hoffmann. "Racing vs. Pacing to Idle: A Comparison of Heuristics for Energy-aware Resource Allocation". In: *HotPower*. 2013.

[20] H. Hoffmann. "CoAdapt: Predictable Behavior for Accuracy-Aware Applications Running on Power-Aware Systems". In: *ECRTS*. 2014.

[21] H. Hoffmann, A. Agarwal, and S. Devadas. "Selecting Spatiotemporal Patterns for Development of Parallel Applications". In: *IEEE Trans. Parallel Distrib. Syst.* 23.10 (2012), pp. 1970–1982.

[22] H. Hoffmann and M. Maggio. "PCP: A Generalized Approach to Optimizing Performance Under Power Constraints through Resource Management". In: *ICAC*. 2014.

[23] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal. "A Generalized Software Framework for Accurate and Efficient Managment of Performance Goals". In: *EMSOFT*. 2013.

[24] H. Hoffmann, S. Misailovic, S. Sidiroglou, A. Agarwal, and M. Rinard. *Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures*. Tech. rep. MIT-CSAIL-TR-2009-042. MIT, 2009.

[25] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. "Dynamic Knobs for Responsive Power-Aware Computing". In: *ASPLOS*. 2011.

[26] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. "Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control". In: *Computers, IEEE Transactions on* 56.4 (2007), pp. 444 –458.

[27] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. "POET: A Portable Approach to Minimizing Energy Under Soft Real-time Constraints". In: *RTAS*. 2015.

[28] E. Kalyvianaki, T. Charalambous, and S. Hand. "Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters". In: *ICAC*. 2009.

[29] E. Kalyvianaki, T. Charalambous, and S. Hand. "Adaptive Resource Provisioning for Virtualized Servers Using Kalman Filters". In: *TAAS* 9.2 (2014).

[30] M. Kambadur and M. Kim. *Energy Exchanges: Internal Power Oversight for Applications*. Tech. rep. CUCS-009-14. Columbia, 2014.

[31] M. Kambadur and M. Kim. "Trading Functionality for Power within Applications". In: *APPROX*. 2014.

[32] M. N. Katehakis and A. F. Veinott. "The Multi-Armed Bandit Problem: Decomposition and Computation". In: *Mathematics of Operations Research* 12.2 (1987), pp. 262–268. DOI: 10.1287/moor.12.2.262.

[33] M. Kim, M.-O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian. "xTune: A Formal Methodology for Cross-layer Tuning of Mobile Embedded Systems". In: *ACM Trans. Embed. Comput. Syst.* 11.4 (Jan. 2013).

[34] E. Le Sueur and G. Heiser. "Slow Down or Sleep, That is the Question". In: *Proceedings of the 2011 USENIX Annual Technical Conference*. Portland, OR, USA, 2011.

[35] W. Levine. *The control handbook*. Ed. by W. Levine. CRC Press, 2005.

[36] B. Li and K. Nahrstedt. "A control-based middleware framework for quality-of-service adaptations". In: *IEEE Journal on Selected Areas in Communications* 17.9 (Sept. 1999).

[37] X. Li, R. Gupta, S. V. Adve, and Y. Zhou. "Cross-component energy management: Joint adaptation of processor and memory". In: *ACM Trans. Archit. Code Optim.* 4.3 (Sept. 2007).

[38] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. "Flikker: saving DRAM refresh-power through critical data partitioning". In: *ASPLOS*. 2011.

[39] M. Maggio, E. Bini, G. C. Chasparis, and K.-E. Årzén. "A Game-Theoretic Resource Manager for RT Applications". In: *ECRTS*. 2013.

[40] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva. "Comparison of Decision-Making Strategies for Self-Optimization in Autonomic Computing Systems". In: *ACM Trans. Auton. Adapt. Syst.* 7.4 (Dec. 2012), 36:1–36:32. ISSN: 1556-4665. DOI: 10.1145/2382570.2382572. URL: http://doi.acm.org/10.1145/2382570.2382572.

[41] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. "Power Optimization in Embedded Systems via Feedback Control of Resource Allocation". In: *IEEE Trans. on Control Systems Technology* 21.1 (2013).

[42] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. "Power management of online data-intensive services". In: *ISCA* (2011).

[43] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann. "A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints". In: *ASPLOS*. 2015.

[44] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling". In: *ICS*. 2002.

[45] K. V. Palem. "Energy aware algorithm design via probabilistic computing: from algorithms and models to Moore's law and novel (semiconductor) devices". In: *CASES*. 2003.

[46] K. V. Palem and L. Avinash. "Ten Years of Building Broken Chips: The Physics and Engineering of Inexact Computing". In: *ACM Trans. Embedded Comput. Syst.* 12.2s (2013), p. 87.

[47] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. "A resource allocation model for QoS management". In: *RTSS*. 1997.

[48] M. Rinard. "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks". In: *ICS*. 2006.

[49] M. C. Rinard, H. Hoffmann, S. Misailovic, and S. Sidiroglou. "Patterns and statistical analysis for understanding reduced resource computing". In: *OOPSLA*. 2010.

[50] E. Rotem, A. Naveh, D. R. amd Avinash Ananthakrishnan, and E. Weissmann. "Power management architecture of the 2nd generation Intel Core microarchitecture, formerly codenamed Sandy Bridge". In: *Hot Chips*. Aug. 2011.

[51] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. "Energy Management in Mobile Devices with the Cinder Operating System". In: *EuroSys*. 2011.

[52] M. Salehie and L. Tahvildari. "Self-adaptive software: Landscape and research challenges". In: *ACM Trans. Auton. Adapt. Syst.* 4.2 (2009), pp. 1–42. ISSN: 1556-4665. DOI: http://doi.acm.org/10.1145/1516533.1516538.

[53] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. "EnerJ: approximate data types for safe and general low-power computation". In: *PLDI*. 2011.

[54] A. Sharifi, S. Srikantaiah, A. K. Mishra, M. Kandemir, and C. R. Das. "METE: meeting end-to-end QoS in multicores through system-wide resource management". In: *SIGMETRICS*. 2011.

[55] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. "Power Containers: An OS Facility for Fine-grained Power and Energy Management on Multicore Servers". In: *ASPLOS*. 2013.

[56] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. "Managing performance vs. accuracy trade-offs with loop perforation". In: *ESEC/FSE*. 2011.

[57] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. "Koala: A Platform for OS-level Power Management". In: *EuroSys*. 2009.

[58] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. “Eon: a language and runtime system for perpetual systems”. In: *SenSys*. 2007.

[59] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. “A Feedback-driven Proportion Allocator for Real-rate Scheduling”. In: *OSDI*. 1999.

[60] SWISH++. http://swishplusplus.sourceforge.net/.

[61] A. Tannebaum. *Modern Operating Systems*. Pearson/Prentice Hall, 2008.

[62] M. Tokic. “Adaptive ϵ-Greedy Exploration in Reinforcement Learning Based on Value Differences”. In: *KI*. 2010.

[63] V. Vardhan, W. Yuan, A. F. H. III, S. V. Adve, R. Kravets, K. Nahrstedt, D. G. Sachs, and D. L. Jones. “GRACE-2: integrating fine-grained application adaptation with global adaptation for saving energy”. In: *IJES* 4.2 (2009).

[64] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. “Server workload analysis for power minimization using consolidation”. In: *USENIX Annual technical conference*. 2009.

[65] M. Weiser, B. B. Welch, A. J. Demers, and S. Shenker. “Scheduling for Reduced CPU Energy”. In: *OSDI*. 1994.

[66] W. Yuan and K. Nahrstedt. “Energy-efficient soft real-time CPU scheduling for mobile multimedia systems”. In: *SOSP*. 2003.

[67] R. Zhang, C. Lu, T. Abdelzaher, and J. Stankovic. “ControlWare: A middleware architecture for Feedback Control of Software Performance”. In: *ICDCS*. 2002.

[68] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. “Survey of Energy-Cognizant Scheduling Techniques”. In: *IEEE Trans. Parallel Distrib. Syst.* 24.7 (2013), pp. 1447–1464. DOI: 10.1109/TPDS.2012.20. URL: http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.20.