

GOAL: Supporting General and Dynamic Adaptation in Computing Systems

Ahsan Pervaiz
University of Chicago
USA
ahsanp@uchicago.edu

Yao Hsiang Yang
Rice University
USA
yao-hsiang.yang@rice.edu

Adam Duracz
Rice University
USA
adam.duracz@rice.edu

Ferenc Bartha
Rice University
USA
barfer@math.u-szeged.hu

Ryuichi Sai
Rice University
USA
ryuichi@rice.edu

Connor Imes
University of Chicago
USA
ckimes@cs.uchicago.edu

Robert Cartwright
Rice University
USA
cork@rice.edu

Krishna Palem
Rice University
USA
palem@rice.edu

Shan Lu
University of Chicago
USA
shanlu@uchicago.edu

Henry Hoffmann
University of Chicago
USA
hankhoffmann@uchicago.edu

Abstract

Adaptive computing systems automatically monitor their behavior and dynamically adjust their own configuration parameters—or *knobs*—to ensure that user goals are met despite unpredictable external disturbances to the system. A major limitation of prior *adaptation frameworks* is that their internal *adaptation logic* is implemented for a specific, narrow set of goals and knobs, which impedes the development of complex adaptive systems that must meet different goals using different sets of knobs for different deployments, or even change goals during one deployment.

To overcome this limitation we propose GOAL, an adaptation framework distinguished by its virtualized adaptation logic implemented independently of any specific goals or knobs. GOAL supports this logic with a programming interface allowing users to define and manipulate a wide range of goals and knobs within a running program. We demonstrate GOAL's benefits by using it re-implement seven different adaptive systems from the literature, each of which has a different set of goals and knobs. We show GOAL's general approach meets goals as well as prior approaches designed for specific goals and knobs. In dynamic scenarios where the goals and knobs are modified at runtime, GOAL achieves 93.7% of optimal (oracle) performance while providing a 1.69× performance advantage over existing frameworks that cannot perform such dynamic modification.

CCS Concepts: • **Software and its engineering** → *Development frameworks and environments*; **Extra-functional properties**.

Keywords: domain-specific language; adaptive computing; control theory; energy; resource allocation

ACM Reference Format:
Ahsan Pervaiz, Yao Hsiang Yang, Adam Duracz, Ferenc Bartha, Ryuichi Sai, Connor Imes, Robert Cartwright, Krishna Palem, Shan Lu, and Henry Hoffmann. 2022. GOAL: Supporting General and Dynamic Adaptation in Computing Systems. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '22), December 8–10, 2022, Auckland, New Zealand.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3563835.3567655

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Onward! '22, December 8–10, 2022, Auckland, New Zealand

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9909-8/22/12…$15.00
https://doi.org/10.1145/3563835.3567655

1 Introduction

Many software systems face the critical challenge of meeting quality-of-service *goals*, expressed as constraints and objectives on *metrics*; e.g., request latency, energy consumption, and result accuracy. As a further complication, these goals must be met despite unpredictable—yet inevitable—runtime

variations in workload and operating environment. To provide predictable behavior in unpredictable deployments, it is crucial to build computer systems that *adapt* by adjusting their configurable components, or *knobs*, as they execute [55, 58].

Implementing adaptive systems requires an *adaptation logic* (AdaptLog) that can efficiently—at runtime—convert observed metrics into knob settings that meet the goals [37, 60, 80, 83, 93]. However, implementing a reliable and robust AdaptLog is difficult. For example, when the hardware for the Samsung Galaxy S9 was upgraded for the S9+, the achieved performance and energy efficiency was worse despite the better hardware [31]. The problem was tracked down to misconfigurations in the AdaptLog of the HMP scheduler [30]. In short, a heuristic-based AdaptLog that was appropriate for one hardware architecture, was inefficient on a closely related, but different architecture.

To ease development of adaptive computing systems, researchers have proposed several *adaptation frameworks* in the form of libraries or language runtimes [10, 15, 20, 21, 25, 33, 49, 51, 54, 56, 74, 96]. Using the framework's interface, developers provide an *adaptation specification* (AdaptSpec): a declaration of the system's goals and the knobs that can be configured to achieve it. The framework's internal AdaptLog then tunes the knobs in response to any runtime changes.

Although helpful, all existing frameworks focus on a narrow, predefined set of possible AdaptSpecs (i.e., one or two metrics and a small collection of knobs). For example, Eon [79] only supports accuracy and energy metrics using alternative method implementations as knobs. Similarly, PowerDial [44] only supports accuracy and throughput tradeoffs using application-level parameters as knobs.

This lack of generality arises because prior adaptation frameworks develop their AdaptLog using specialized *models* that relate specific metrics to specific knobs. Whether the AdaptLog is based on machine learning, control theory, or heuristics, the model is essential to predict how metrics will change with changes in knobs, which then guides the AdaptLog to set the knobs to ensure the goals are met. However, because the model relates specific knobs to specific metrics, the relevant knobs and metrics need to be enumerated before the model is constructed and that model must be reconstructed for use with a different knobs and/or metrics. This reliance on a narrowly defined model makes it difficult to implement a *general* adaptive system that can deploy with different goals in different environments.

The use of fixed models also prevents a system from *dynamically* changing AdaptSpecs during execution. We define the runtime alteration of an AdaptSpec as *meta-adaptation*. For many applications it is not enough to just adjust knob configurations; meta-adaptation is necessary as the goals themselves must be changed in response to external conditions [35, 71]. For example, consider a CCTV camera installed with a backup battery [76]. It must always meet a target frame rate to prevent data loss, but its other goals vary depending on power source. On line power the system must maximize quality, however, during a power outage, it must minimize energy to prolong battery life [1]. Running this system in either AdaptSpec for its entire execution is suboptimal, either wasting energy on battery or lowering quality on line power.

To support more general and dynamic adaptation, including meta-adaptation, this paper presents GOAL: Goal-Oriented Adaptation Language. GOAL provides novel adaptation framework implemented in Swift [5]. Its key components are its (1) runtime AdaptLog and (2) its interface for writing AdaptSpecs.

Central to GOAL's design is its AdaptLog, which takes the form of a *virtualized, time-variant, adaptive* control system. Unlike prior approaches, GOAL's *virtual* control system is independent of any specific model relating metrics to knobs; instead, it is parameterized by a model which is passed in at runtime. Furthermore, GOAL's controller continually adjusts itself at runtime while also carefully exploiting structure of optimization problems so that it can control non-linear systems with a series of linear approximations.

GOAL's AdaptSpecs are written using a novel domain specific language (DSL), which is compiled just-in-time (JIT), separating the AdaptSpec declaration from system implementation. This separation allows different AdaptSpecs to be used with the same binary for deployments with different requirements or even for changing the requirements while the system is running. GOAL also provides a Library API so users can declare knobs and metrics and alter these values during execution. These features support complex adaptive behavior that would have been difficult and inefficient to implement with existing frameworks.

To demonstrate GOAL, we re-implement seven adaptive applications from the literature. Collectively, these case studies cover a wide range of metrics (throughput, latency, accuracy, power, cost, reliability and efficiency) and knobs (at both the application and system level, including two different hardware systems with distinct knobs). Our results show that GOAL's generalized approach meets goals just as well as prior work that synthesizes AdaptLogs specifically for each application's narrow goals and knobs [26]. To highlight GOAL's benefits, we then modify each application to perform meta-adaptation. We observe that due to GOAL's ability to support a wide range of AdaptSpecs and meta-adaptation, GOAL-based applications exhibit a 1.69× average improvement in corresponding metrics after meta-adaptation is performed, compared to prior approaches that cannot support meta-adaptation. Furthermore, we show that GOAL incurs negligible overhead and is robust to errors in profiling, changing workloads and operating conditions.

This paper makes the following contributions:

- Motivates the benefits of support general purpose adaptation and meta-adaptation.

- Proposes a general adaptation logic and runtime that supports a wide range of knobs, metrics and goals.
- Proposes a programming framework and DSL for writing adaptation specifications.
- Implements GOAL and releases it as open source.[1]

2 Related Work and Motivation

Adaptation is a key mechanism for building robust software systems that operate effectively despite unpredictable dynamic changes to inputs or operating environment [55, 58]. This section discusses prior approaches to building adaptive software systems, and discusses how limitations of prior works motivate the need for adaptation as a first class programming construct.

2.1 Complexity Requires Adaptation

Computer systems have numerous configuration parameters and settings that impact their ability to meet their quantifiable behavioral goals [55]. Improper configuration is a notorious source of performance issues and bugs [47, 69, 90]. Selecting a good configuration is difficult because optimal configurations depend upon dynamically varying external factors such as workload and operating conditions [36, 59, 87]. Adaptive systems address this problem by automatically and dynamically adjusting configuration parameters to ensure goals are met. Thus, there is a need for principled approaches to building adaptive computing systems recognized by both industry [11, 32, 39, 45, 55, 58, 67] and academia [24, 29, 46, 68].

Two design patterns—Observe-Orient-Decide-Act (OODA) [12, 13, 73] and Monitor-Analyse-Plan-Execute (MAPE) [55]—have been proposed for creating adaptive software. Both establish a *control loop* as the basic structure for adaptation. During a loop iteration the software first *observes/monitors* its environment and its own quantifiable behavior. It then *orients/analyzes* itself with respect to these metrics to *decide/-plan* what should be done next. The subsequent iteration then *acts/executes* these decisions by changing the values of configuration parameters. Because it results in more robust and flexible software, many approaches implement adaptive loops in the OODA/MAPE pattern.

2.2 Existing Support for Adaptation

As mentioned earlier, prior work models the OODA/MAPE design pattern as a control loop and several researchers have proposed that general scheme as a basis for system [42], software [8], and language [72] design. Many existing works suggest control theory [7, 27, 28, 40, 62, 75, 77, 81, 88, 94, 95], machine learning [9, 22, 38, 50, 91], and combinations of the two [41, 43, 57, 65, 85] as the basis for building principled *AdaptLogs* that perform the orientation/analysis and decision/planning phase of the OODA/MAPE loop to dynamically adjust configuration knobs in a running computer system. Such approaches provide formal, mathematical guarantees about the precise assumptions and operating conditions under which the goals will be met. However, a challenge is that specialized knowledge in control, learning, or both is required to successfully deploy such approaches.

To make principled adaptation easy to implement, recent work proposes *adaptation frameworks* that package a control- or learning-based AdaptLog into a programming language [4, 6, 10, 15, 16, 54, 56, 74, 79, 96] or a library [19, 21, 29, 33, 51, 65, 70, 78, 84, 90, 95] runtime. Developers do not need to possess specialized knowledge in learning or control to use these frameworks. Instead, they instantiate it with an initial AdaptSpec, after which the AdaptLog monitors the metrics and independently tunes the knobs to meet the goal.

While the OODA/MAPE design patterns themselves provide a general strategy, their implementations in existing approaches have significant limitations. These limitations arise because each framework is designed to support specific metrics and knobs and their AdaptLogs do not generalize to the metrics and knobs in other works. In other words, the OODA/MAPE concepts are generalizable, but specific instantiations of these concepts are problem-specific. Table 1 illustrates this idea, showing that while there is wide support for different goals and knobs across frameworks, the support provided by any one is specific and thus limited. For example, Green uses a heuristic AdaptLog based on a specific model of how alternative function implementations affect an application's power and accuracy tradeoffs [6]. Similarly, Aeneas's reinforcement learning model uses a reward signal based on energy measurements, and introducing new metrics requires a new reward function and learning model. To the best of our knowledge, there is no single framework that generalizes across a wide range of goals and knobs. This limitation also means that existing frameworks cannot be used for meta-adaptation because they do not support any alternative AdaptSpecs.

Overcoming these limitations requires users to extend the framework's internal AdaptLog for additional AdaptSpecs. This entails reconstructing the model and reimplementing the AdaptLog and its interface to support other knobs, metrics, goals and meta-adaptation. However, doing so defeats the purpose of using an adaptation framework because it requires users to have specialized knowledge of the AdaptLog.

While prior work has also explored making adaptation components configurable [3], we believe that performing meta-adaptation using such works is difficult because such works encapsulate the application itself rather than making the adaptation framework a natural component of the application which would allow it to exert fine grained control on all aspects of adaptation.

[1] GOAL source code available at: https://github.com/GOAL-Adaptation.

Table 1. Comparison of Selected Adaptation Frameworks

Framework	Supported Goals		Supported Knobs	Runtime Modifications
	Constraint Metric	Objective Function		
Aeneas [4]	App. level constraint	Energy	App. parameters	N/A
SiblingRivalry [4]	Throughput	Power	Alternate func. implementations	Constraint target value
Green [6]	App. level constraint	Power or Accuracy	Code approximation	N/A
Eon [79]	Power	Accuracy	Alternate func. implementations	N/A
JouleGuard [41]	Energy	Accuracy	App. and Sys. parameters	N/A
Truffle [25]	App. Level constraint	Energy	Code approximation	N/A
Odyssey [29]	Accuracy	Energy	App. alternatives and Sys. parameters	N/A
GOAL (this paper)	**Any**	**Any**	**Any first-class object**	**Goals, Metrics, and Knobs**

Thus, we argue that the aforementioned limitations would be best addressed through a generalized adaptation framework that allows applications to interact with all aspects of adaptation dynamically. However, developing such a framework is not trivial because it requires a uniform interface and an AdaptLog whose underlying model allows the declaration and use of AdaptSpecs that work with any user-defined measures, knobs and goals. Furthermore, to enable meta-adaptation, the framework must coordinate the interface with the runtime to ensure that goals are met even when the AdaptSpec changes. This paper introduces such a framework, GOAL.

3 Implementing Adaptation with GOAL

We provide a high-level overview of GOAL, later sections detail its AdaptLog (§ 4) and interface (§ 5). We implement a meta-adaptive video encoder for a CCTV camera that meets a target frame rate, while maximizing quality on line power and minimizing energy on battery. Figure 1 shows this example: the original Swift code without adaptation (a), the GOAL version (b), and an example GOAL AdaptSpec (c).

3.1 Original (Non-adaptive) Code

As shown in Figure 1a, the developer initializes the encoder and defines parameters: `qp` and `me` (quantization parameter and motion estimation algorithm, respectively), which govern tradeoffs between the video quality and frame rate. Subsequently, the system enters a while loop where it calls the `encodeNextFrame` method with `qp` and `me` as input. The `record` function logs per-frame encoding quality.

The code in Figure 1a neither meets a target frame rate, nor reacts to changes in power source. Carefully selecting static values for `qp` and `me` could ensure the frame rate; however, without adapting to frame-to-frame differences, quality will be sacrificed (e.g., set the parameters for the worst case and live with lower quality for other cases). In fact, video encoding is a challenge problem for adaptive computing because it is difficult to tune encoding parameters [63].

3.2 Adding Adaptation with GOAL

GOAL provides constructs that allow developers to implement all parts of the OODA/MAPE control loop with minor modifications to existing software. Concretely, the developer needs to: (1) identify and declare configurable components as `Knobs`, (2) specify the code segment in which to perform adaptation, and (3) report application-level metrics to the GOAL runtime using `measure`.

Figure 1b illustrates the CCTV system implemented with GOAL. `qp` and `me` are declared as `Knobs`, using a stringified name and a default value for initializing the knob. The relevant metrics to be monitored are identified using `measure`. For example the encoding `quality` is reported for the CCTV. The `while` loop is replaced with GOAL's `optimize` loop (§ 5.1) whose inputs are the list of `Knobs` to tune, and the loop body. This syntax tells the GOAL runtime to iteratively execute the loop body (as in the original program) while tuning the knobs after each iteration, according to the AdaptSpec. Hence, with such minor modifications we have converted the non-adaptive application to an adaptive application that implements a complete OODA/MAPE control loop.

3.3 Writing Adaptation Specifications

The next step is writing an AdaptSpec. Figure 1c shows an example, defining the `knobs`, allowed values for each, the `measures` to monitor, and the *goal* that formally specifies the constraints and objective. This example is how the CCTV should behave on line power: maximize quality, with a throughput of 30 iterations per second (which translates to 30 frames/s). The application-level metrics (e.g., quality) and knobs (e.g. `qp` and `me`) in the AdaptSpec must match the string arguments in calls to `Knob` and `measure`. GOAL's runtime automatically declares metrics such as throughput, energy, etc. and hardware specific knobs such as core count (`numCores`) and DVFS frequency (`coreFreq`), allowing them to be used without explicit declaration.

This AdaptSpec tells the runtime to measure quality, energy, and throughput, while tuning `qp`, `me`, `numCores` and `coreFreq`. Additionally, the AdaptSpec states that the runtime must choose configurations that meet a further constraint on the knobs: the product of `numCores` and `coreFreq` should be greater than 2400. See § 5.3 for a full description of the DSL's grammar and capabilities.

GOAL AdaptSpecs are compiled just-in-time (§ 4.2) and can be written in text files independent of the application code. This allows developer's to produce a single binary that

```
let encoder = initEncoder()
var qp = 30
var me = 1

while(true)
{

  encoder.encodeNextFrame(qp,
    me)
  record(encoder.getQual())


}
```

(a) Original Code

```
import GOAL
let encoder = initEncoder()
var qp = Knob("qp", 30)
var me = Knob("me", 1)

optimize("encoder", [qp, me])
{

  encoder.encodeNextFrame(qp.get(),
    me.get())
  measure("quality",
            encoder.getQual())

}
```

(b) GOAL Code

```
goal encoder
  max(quality)
    such that throughput == 30.0
measures
  quality: Double
  energy: Double
  throughput: Double
knobs
  qp = [10, 30 reference]
  me = [1 reference, 5]
  coreFreq = [600, 1200 reference]
  numCores = [2, 4 reference]
such that
  numCores * coreFreq > 2400
```

(c) Adaptation Specifications

Figure 1. The encoder application.

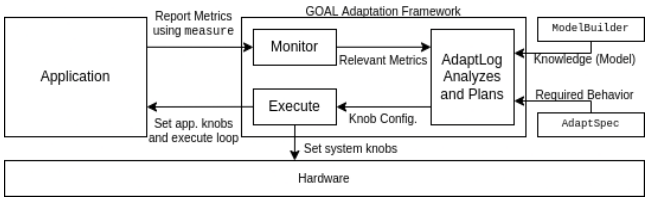

Figure 2. The GOAL Adaptation Framework.

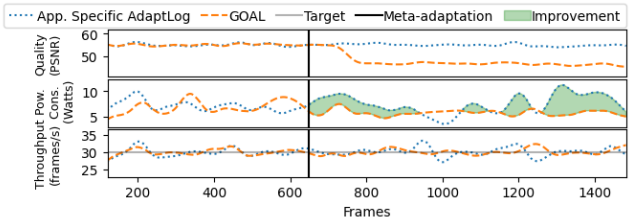

Figure 3. GOAL meets all CCTV requirements robustly.

can be deployed to meet different goals by writing different AdaptSpecs. Such flexibility distinguishes GOAL from existing adaptation frameworks.

Given the GOAL system and an AdaptSpec, the last step before deployment is to use GOAL's model builder (§ 4.3), which runs the application on test inputs to learn a function the knob configurations' impact on the specified metrics. Figure 2 presents a high-level overview of the OODA/MAPE loop implemented using GOAL. During execution, GOAL monitors the metrics identified by `measure`, analyzes and plans which values to use for each of the Knobs to meet the goal specified in AdaptSpec, then sets the Knobs to those values and executes the loop body.

Finally, the system can dynamically change any aspect of the AdaptSpec during execution, telling GOALs runtime to meet new goals, use new knobs, or both.

3.4 Adding Meta-Adaptation

The CCTV requires meta-adaptation to change goals based on power supply. GOAL's `intend` method uniquely supports this by dynamically changing the AdaptSpec in a running system. The `intend` function takes a string representation of the goal and replaces the active goal with this argument. To illustrate this, we augment the `optimize` loop in Figure 1b with the following code (not shown in the figure):

```
if getPowerSupply() == .DirectPower  {
  intend(to: .maximize, objective: "quality",
    suchThat: [(measure: "throughput",
                goal: 30.0)])
} else if getPowerSupply() == .Battery {
  intend(to: .minimize, objective: "energy",
    suchThat: [(measure: "throughput",
                goal: 30.0)])
}
```

The code in red specifies the parameters of `intend` and the code in blue represents the arguments specific to the requirements of the CCTV system.

With this handful of changes, the encoder will now meet the throughput constraint while optimizing either quality or energy based on the power source. This large increase in adaptive capability for small code changes highlight how GOAL's general adaptation framework supports complex adaptive behavior and seamless meta-adaptation.

3.5 Quantitative Benefits of Using GOAL

Figure 3 compares the execution of our GOAL CCTV with a version that uses prior work to synthesize a customized, Application-specific AdaptLog [26]. However, using prior work, the CCTV can only meet one goal; we begin with the goal corresponding to line power: maximize quality and meet a throughput constraint.

Both versions execute identically until a power outage (at frame 650), where the GOAL version performs meta-adaptation to start reducing energy. In contrast, the Application specific version cannot change the goal, continues using high energy, and risks depleting the battery, rendering the camera inoperable. After frame 650, GOAL consumes

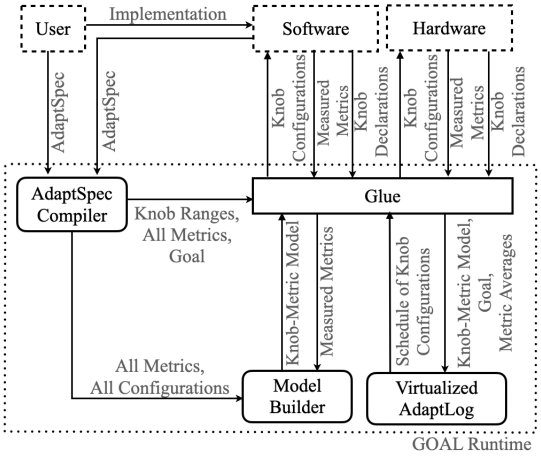

Figure 4. The GOAL Runtime.

32% less energy than the Application-specific approach. An additional AdaptLog could have been synthesized to run the CCTV with minimal energy. However, such a version would needlessly sacrifice quality while operating on line power, and switching between independent AdaptLogs at runtime is quite costly—in both execution time § 7.3 and engineering effort § 7.4. This example shows the importance of providing support for general adaptation and meta-adaptation and also the ease with which this can be achieved using GOAL.

4 The GOAL Runtime

Figure 4 illustrates GOAL's runtime, which consists of three key pieces: (1) the virtualized AdaptLog, (2) the AdaptSpec Compiler, and (3) the Model Builder. The Glue code coordinates these three components with the rest of the software and hardware. Furthermore, some application and system level metrics such as throughput and power consumption are measured by the Glue code [48]. The AdaptLog (§ 4.1) is implemented independently of any specific AdaptSpec; it receives the goals, the model, and current metrics and produces a *schedule* of configurations. The compiler (§ 4.2) takes an AdaptSpec and produces four outputs: (1) the space of allowable knob configurations, (2) metrics to monitor, (3) any constraints on knobs and (4) the goal to meet. The goal is defined as a *constrained optimization problem* (COP) in GOAL's DSL (§ 5.3). Before deployment, the model builder (§ 4.3) samples configurations from the compiled AdaptSpec and executes the system in those configurations while observing the metrics listed in the AdaptSpec. From these observations the model builder learns a model that estimates changes in metrics as a function of knob configuration. During deployment, the AdaptLog schedules knob configurations to optimally meet the goal according to the model's estimates.

4.1 A Virtualized Adaptation Logic

GOAL virtualizes a control theoretic adaptation logic using two key principles: *relativity* and *translation*. While typical control systems find absolute values for the knobs they configure, GOAL controls virtual values that represent the necessary change in behavior—*relative* to the default knob configuration—that must be achieved to meet the goals. This virtual value by itself is not useful; it must be *translated* into a schedule of specific knob configurations. Critically, this translation can be done online once the metrics and knobs are known and the Model Builder (§ 4.3) has produced the function that estimates metrics given knob configurations.

4.1.1 A Traditional Control Example.

Filieri et al. propose a framework for automatically synthesizing controllers for software systems [26]. We use this approach to illustrate a typical way that AdaptLogs are created, specifically building a controller for the qp knob from the CCTV example (§3). We begin assuming the original goal: meeting a frame rate constraint with maximum quality. Following [26], we first learn a simple linear model, relating frame rate to qp:

$$\text{FPS}(t) = 0.354 \cdot \text{qp}(t-1) \tag{1}$$

The constant 0.354 is specific to *qp*. To meet a target performance, the controller monitors the current performance $\text{FPS}(t)$ at time t and computes the error with the desired performance FPS_{goal}: $e(t) = \text{FPS}_{goal} - \text{FPS}(t)$. With this error and the model from Equation 1, we set qp at time t as:

$$\text{qp}(t) = \text{qp}(t-1) - \frac{e(t-1)}{0.354} \tag{2}$$

Equation 2 is a simple and efficient AdaptLog, suitable for qp. However, this AdaptLog is not general; it is entirely specific to (1) the constraint metric (frames per second in this example) and (2) the available knobs (qp).

While this example uses a control approach, similar problems occur with other techniques. For example, reinforcement learning (RL) is a popular basis for AdaptLogs [16, 64, 66, 86, 89]. However, RL requires a reward function and a set of possible actions; in existing frameworks, the reward is tied to specific metrics, while the actions are tied to sets of specific knobs (e.g., application alternatives for Aeneas [16]). We are not aware of a way to virtualize the actions in RL based AdaptLogs that is both useful for optimization and general with respect to user-defined knobs.

4.1.2 GOAL's Virtualized Control Logic.

We now show how subtle changes in the above formulation implement a virtual control signal. The key here is to model *relative* behavior rather than absolute metrics as in typical control approaches. We then *translate* that relative behavior into specific knob settings. This formulation provides a layer of indirection. The controller (which only understands relative values) can be implemented independently of any specific knobs or metrics. The additional logic for translating a relative value into specific knobs settings is parameterized by the learned model that enables the translation (§ 4.1.4).

We begin by noting that a simple equation relates the behavior in any metric m at time t to a scalar multiple $xup(t-$

1) of some baseline behavior m_{base}:

$$m(t) = m_{base} \cdot xup(t-1) \tag{3}$$

Given this relationship, to control metric m, we first compute the error between the target behavior (m_{target}, the constraint from the AdaptSpec) and the measured behavior ($m(t)$):

$$e_m(t) = m_{target} - m(t) \tag{4}$$

We can then control the behavior by tuning the $xup(t)$:

$$xup(t) = xup(t-1) - \frac{e_m(t-1)}{m_{base}} \tag{5}$$

Equation 5 looks similar to Equation 2, but instead of a constant appropriate for one knob, Equation 5 is parameterized by the base behavior for this metric; i.e., the metric's expected value in the default knob configuration.

While independent of any specific knobs, this approach is clearly dependent on the base behavior m_{base} for the application in metric m; i.e., the expected metric value when the application's knobs are all set to their default values. This value will obviously vary from application to application and even as the application runs.

4.1.3 Adapting to Workloads.

To adapt the controller to the current behavior in metric m at runtime, GOAL's AdaptLog continually estimates m_{base}, using a Kalman Filter [92], an approach used in prior work [52, 53]. Thus, if the behavior varies during execution, this estimation compensates and ensures that the constrained metrics can still be controlled. This process is analogous to approximating a nonlinear function (in this case the application's behavior with respect to execution time) with a series of tangent lines, where m_{base} is the tangent's slope. More formally, GOAL's Kalman filter estimates the base behavior at time t as $m_{base}(t)$ and uses this estimate in place of m_{base} in Equation 5. GOAL uses a standard Kalman filter formulation:

$$\begin{aligned}
m^-_{base}(t) &= m_{base}(t-1) \\
e^-_b(t) &= e_b(t-1) + q_b(t) \\
k_b(t) &= \frac{e^-_b(t) \cdot xup(t)}{xup(t)^2 \cdot e^-_b(t)} \\
m_{base}(t) &= m^-_{base}(t) + k_b(t)\left(\frac{1}{m(t)} - s(t) \cdot m^-_{base}(t)\right) \\
e_b(t) &= [1 - k_b(t) \cdot xup(t-1)]e^-_b(t)
\end{aligned} \tag{6}$$

In this formulation, $k_b(t)$ is the Kalman gain for the constrained metric, m, being controlled. The $m(t)$ denotes measured behavior of the constrained metric during the last window. The $m^-_{base}(t)$ and $m_{base}(t)$ are the next to last and last estimates of m_{base}. Similarly, $e^-_b(t)$ and $e_b(t)$ are the next to last and last estimates of the error variance.

The Kalman Filter is useful because it provides optimal estimates of the application workload and is exponentially convergent [17]; i.e, the estimate will converge in a number of iterations proportional to the logarithm of its error. Furthermore, the user or the developer does not need to provide any additional data, these guarantees are provided using data that is available to GOAL during execution.

4.1.4 Scheduling Knob Configurations.

Even after accounting for workload changes, the virtual xup from Equation 5 is not useful by itself; it must be *translated* into actual knob configurations. A set of values for knobs can be represented as a vector k, and a knob configuration is an assignment of a value to each knob component in the vector. For example, our video encoder from Figure 1 has a knob vector with four components (one each for `qp`, `me`, `numCores` and `coreFreq`), and the default configuration is $\langle 30, 1, 1200, 4\rangle$. Each knob configuration has an expected effect on each metric, as estimated by GOAL's model builder. In our example those metrics are throughput (frame rate), quality, and energy. In addition, the xup signals are continuous while the available knob settings are discrete.

We translate xup from Equation 5 to discrete knob configurations by *scheduling* over time; i.e., spending different amounts of time in knob configurations such that the average over the time period is the desired continuous value. This scheduling problem is formulated as a constrained optimization problem (COP, which is extracted from the AdaptSpec by GOAL's compiler) where the xup value is the constraint to be met and the decision variables are the time to spend in the knob configuration vectors:

$$\text{optimize} \sum_T F(k) \cdot T_k \tag{7}$$

$$\text{s.t.} \quad xup(t) = \frac{1}{T} \cdot \sum_k \hat{xup}_k \cdot T_k \tag{8}$$

$$T = \sum_k T_k \tag{9}$$

$$T_k \geq 0, \forall k \tag{10}$$

Here T is the time window (defaults to 40 but can be set using an environment variable) over which to schedule, T_k is the time to spend in the kth knob configuration, and $\hat{xup}_k$ is the expected xup for k (from GOAL's model builder §4.3). $F(k)$ is an objective function over knobs defined in the AdaptSpec and extracted by the compiler. For example, in the CCTV application (§ 3), the objective on line power is to maximize quality. Equation 8 requires that the average of all configurations' predicted $\hat{xup}_k$ values come out to the desired $xup(t)$ value, while Equation 9 requires that the sum of times spent in each configuration is equal to the total time over which we are scheduling. Equation 10 ensures that there are no negative time values. The controller sets the virtual $xup(t)$ to ensure the AdaptSpec's constraint is met, and this optimization problem ensures that virtual signal is translated into specific knob settings to deliver the desired xup.

4.1.5 Handling Non-Linear Behavior.

GOAL's AdaptLog uses a linear control model (Equation 5) and solves a

linear optimization problem (Equations 7–10). Of course, most computer systems will exhibit non-linear behavior, especially when combining knobs. Thus, we discuss why this formulation suffices to handle non-linearities. Note the adaptive control (§ 4.1.3) approximates non-linear shifts in application behavior over time. We therefore focus on non-linear interactions in knob behavior. For example, the performance of a knob configuration is likely a non-linear function of each component knob in the configuration. The key intuition is that Equations 7–10 transform a non-linear optimization over the space of all knobs into a linear problem over an exponential search space. Fortunately, however, the problem structure makes it practical to solve.

GOAL's model builder estimates the $\hat{xup}_k$ values for each knob configuration. Given that k here is a vector of knob configurations, the total size of the configurations space is the cross product of all allowable knob settings. So, while the optimization problem in the previous section is linear, there is a nonlinear number of decision variables (the times to spend in each configuration).

However, there are only two non-trivial constraints (Equations 8 and 9). By the duality of optimization problems there is an optimal solution where exactly two of the configurations are allocated non-zero time [14]. Furthermore, those two configurations correspond to two configurations on the convex hull of the tradeoff space represented by the values of F and xup. For example, if the goal is to meet a performance constraint (xup) and optimize accuracy (F), then the optimal solution will involve two configurations on the optimal frontier of performance and accuracy; specifically, the two configurations whose estimated $\hat{xup}_k$ values are just below and just above the target $xup(t)$ [14]. Those two configurations can easily be found from a lookup table, so GOAL's Glue takes the model from the model builder and the AdaptSpec from the compiler and forms a lookup table that considers only the Pareto-optimal tradeoffs in the objective and constraint metrics. Sorting into the Pareto-optimal points can still be expensive ($O(|k| \log(|k|))$, but is only done when a new AdaptSpec is made available. We evaluate the practical overhead in § 7.3 and § 7.8.

4.1.6 Control Theoretic Formal Properties. The most important guarantee is that the system converges to the constraint. Unlike AdaptLogs based on learning methods such as RL or Bayesian Optimization which do not provide guarantees of convergence to constraint, GOAL's AdaptLog uses adaptive control. It thus inherits the formal control theoretic guarantees of a typical control system [40]. GOAL's AdaptLog will converge provided that the estimated base behavior $m_{base}(t)$ and the predicted speedups $\hat{xup}_k$ are within a factor of 2 of their true values. This analysis is based on straightforward application of control theory [26]. Furthermore, even if the base behavior estimation is incorrect momentarily, the Kalman filter is exponentially convergent in error, meaning that estimate will be corrected in a logarithmic number of iterations. We evaluate GOAL's empirical error tolerance (§ 7.6) and find that GOAL's practical behavior matches the theoretical analysis: GOAL tolerates extremely large overestimations of xup and tolerates under-estimations up to a factor of 0.55; e.g., predicting 16.5 frames per second when the true behavior is 30. This robustness is yet another reason we favor a control theoretic formulation for GOAL's AdaptLog and it is consistent with prior work that shows adaptive control techniques do a better job of meeting operating constraints than learning-based techniques [61].

4.1.7 Connecting with the Rest of GOAL. GOAL's virtualized adaptation logic requires the following parameters to be supplied by the rest of the GOAL runtime:

- **Target behavior m_{target}:** Users specify this value as a constraint in their AdaptSpec and it is extracted by the JIT compiler (§ 4.2).
- **Objective function F:** This too is provided by the user in the AdaptSpec, extracted by the compiler.
- **Expected behavior for target metrics ($\hat{xup}_k$):** These values are stored in lookup tables, dynamically constructed by the runtime using the model learned using the model builder (§ 4.3).
- **Schedule window (T):** This value is the `window` parameter to GOAL's `optimize` method. T is provided to the AdaptLog at initialization.
- **Measured Behavior ($m(t)$):** These measurements are provided by the runtime.

This parameterized AdaptLog is general and dynamic. It is general because it works with any knobs and measures that can be specified using GOAL. It is dynamic because the runtime can rapidly change the AdaptLog by simply passing in new parameters.

4.2 Adaptation Specification Compiler

The GOAL compiler extracts the following from the AdaptSpec: (1) a list of metrics, (2) a list of all knob configurations, (3) knob constraints and (4) the goal, which includes m, m_{target} and F (§ 4.1.7). All values except for F are used to initialize the GOAL runtime and interpret the model learned by the model builder. However, F needs to be evaluated repeatedly when computing schedules as it is the objective function to optimize. F cannot be precomputed because it might be an arithmetic expression over several metrics. Therefore, we adopt a two-step evaluation approach.

In the first step, all aforementioned objects from the AdaptSpec's abstract syntax tree are translated to Swift objects using the technique proposed by Carrette et al [18]. All configurations from the extracted list are converted into *knob configurations objects*. A *knob configuration* is an object with an `apply` method, that sets the application and system level

knobs to their corresponding values. These objects correspond to the k from Equations 7–10. The m and knob constraints are used to make a lookup table containing the $\hat{xup}_k$ and a configuration object for configurations, k, that meet all knob constraints. F is translated into a Swift closure so that it can be executed by the AdaptLog with low overhead while computing schedules. In the second step, the AdaptLog continually executes the aforementioned closure—once the observed values for each `measure` are available—to evaluate F as it produces the knob schedule. During execution, a GOAL application can modify the AdaptSpec which triggers recompilation (starting over from the first step).

4.3 Model Builder

GOAL's model builder operates in two modes. In the first—pre-deployment—it runs user-specified test workloads and measures how changes in knobs cause changes in metrics. Specifically, it executes a sampled subset of all knob configurations and estimates the metrics for all allowable knob configurations. The model builder then learns the model using this collected data by using linear regression to compute a piecewise linear model representing the expected behavior for a metric given a knob configuration. In the second mode—during deployment—the model builder interprets and uses the learned model to predict the impact of particular knob configurations on all metrics. Prior work could be used to replace our learner with more accurate or efficient learners [20, 21, 23]. However, our empirical evaluation shows that GOAL's control system is robust to substantial errors in profiling (§ 7.6), so we use piecewise linear models as a proof of concept and leave the investigation of more advanced learning methods to future work.

4.4 Limitations

While GOAL provides significant advantages over prior work, there are several aspects that deserve further discussion.

Foremost, GOAL can only manage goals that can be expressed in terms of quantifiable measures that can be used by the adaptation logic to make adaptation decisions.

GOAL's AdaptLog handles non-linear, non-convex optimization problems by exploiting problem structure and Pareto-optimality to schedule combinations of knob configurations (§ 4.1.5). This approach works because there are a small number of possible constraints. Essentially, we have transformed a problem that would be exponential in the space of possible knob configurations into a problem that is exponential in the number of constrained metrics. In practice, we believe this is a reasonable tradeoff because there are typically many knobs that affect one constraint and there is usually only a small number of metrics for which there is required behavior. Thus, the number of constraints will be much, much smaller than the number of possible knob configurations.

GOAL actuates hardware-specific knobs on the system's behalf. While it is easy to manipulate some hardware platform specific knobs (e.g., the core usage), others (e.g., DVFS frequency) require special permissions. Hence, GOAL systems may require elevated privileges to manipulate certain knobs. § 7.5 shows that system and application knobs can be split into multiple modules, however.

As a proof of concept, GOAL's semantics currently only support discrete knobs. Interesting future work would be assessing the benefits of continuous knobs.

5 The GOAL Programming Interface

We implement GOAL as an extension to Apple's Swift [5]. GOAL consists of ~9.5K lines of code written in a combination of Swift and C, not including third party libraries.

5.1 The GOAL Library API

At a high-level, The GOAL library provides a type (`Knob`), and two functions (`measure` and `optimize`) to create an adaptive application. GOAL provides three additional functions: `restrict`, `control` and `intend` to facilitate meta-adaptation. Together these functions permit dynamic modification of the entire AdaptSpec. The semantics of `Knob`, `optimize` and `intend` are described in the example (§ 3.2).

To facilitate meta-adaptation, `intend` can be used to modify goals, as shown in the CCTV example. Other scenarios might require manipulating knobs and knob ranges, so GOAL's `Knob` type also provides `restrict` and `control` methods. `restrict` explicitly defines a range of values for a particular `Knob`. The runtime uses this range to constrain the configuration space available for adaptation. Calling the method without any arguments fixes the `Knob` to the value it had at the time of the method call. The `control` method removes any limits from previous calls to `restrict`.

Table 2 provides a more detailed description of the GOAL library API.

5.2 Supporting Multi-Module Adaptation

In large computing systems, multiple developers might want to independently develop adaptive modules and GOAL supports this by allowing multiple `optimize` calls. However, the sets of knobs used by different `optimize` calls have to be disjoint and the GOAL runtime will throw an error if this is not the case. After initialization, the runtime manages execution and actuates knobs of each `optimize` independently.

During execution, one module might modify a knob that affects a metric monitored by another module. For example, a module might meet a power goal by lowering clock frequency and thus reduce throughput in a different module. The base speed of the latter module will thus be underestimated. However, GOAL's adaptive control (§ 4.1.3) will observe and account for this change. In this example, the runtime's base speed estimation will update to account for

Function/Method Prototype	Description
`Knob(name: String, initValue: Double)`	Constructor for a knob object that will be actuated during execution to meet specified goal.
`Knob::get()`	Method to retrieve the current value of the knob.
`Knob::restrict(values: [Double]? = nil)`	Method to tell the runtime to only use values from the provided `values` argument to meet the goal. If no `values` is provided, then the knob is fixed to value from when the method was called.
`Knob::control()`	Method to remove all value restrictions for the knob imposed using `Knob::restrict`.
`measure(name: String, value: Double)`	Function to report the observed value of a metric to the runtime. The stringified name is required to identify metric for which the value is being reported.
`optimize(name: String, knobs: [Knob], loopBody: Fn())`	Function to initialize and repeatedly run code given in loopBody while actuating given knobs to meet the goal. In case of multiple optimize blocks, each is given a unique name.
`intend(to: Enum[.minimize | .maximize], objective: String, suchThat: [(measure: String goal: Double)])`	Function to modify the active goal. Using the function we tell the runtime to maximize or minimize the `objective` while meeting the constraint provided using the `suchThat` argument.

Table 2. GOAL Library API Overview

```
goal LoopName
  OptimizationType(Objective) such that
    Constraint == Target
measures
  MetricList
knobs
  KnobList
such that
  KnobExpression < | <= | == | >= | > Expression
```

Figure 5. GOAL Adaptation Specification Language Semantics

the lowered frequency. GOAL will meet goals of all modules if they are noncompeting (i.e., GOAL cannot constrain system power in one module and minimize it in another). Supporting competing goals is an open research problem in adaptive computing. Having language support could make this problem easier and we leave that to future work.

5.3 Adaptation Specification Language

Figure 5 shows the general template and syntax of GOAL's adaptation specification language. Concretely, An AdaptSpec consists of a constrained optimization problem (COP), the metrics to monitor and the knobs along with their valid values that can be used to meet the COP. Finally, it may also contain an optional section that defines additional constraints on the knob values.

Goal This section encodes a COP, expressing required application behavior in terms of its metrics. The goal has five parts:

- The `OptimizationType`, can be either `min` or `max`.
- The `Objective`, an expression on `measures`.
- The `Constraint`, a metric (`latency` in our example) that is associated with constraints.
- The `Target`, the constraint value (`30.0` seconds per iteration in our example).

Measures This section declares quantifiable metrics like latency, bitrate, etc., that should be observed by the runtime. The metrics are declared as a `MetricList` which is a newline separated list of the form '`metric name: datatype`'. Currently, metrics datatype may only have the `Double` type, but any totally ordered type that supports the operations used in the objective function could work.

Knobs This section defines the available *configuration space* as `KnobList` which is a newline separated list of the form '`knob name = [ values ]`'. Each entry is a `knob` definition which consists of a name, a *range* expression that evaluates to a list of constants, and a reference value. The name associates the knob definition with a `Knob` instance in the application. The list of `Knobs` in the AdaptSpec can be a subset of the `Knobs` from the code. The runtime will only use the `Knobs` defined in the AdaptSpec during execution. For all the `Knobs` not defined in the AdaptSpec, the runtime will only use the reference value in the code. Reference values in the AdaptSpec override those passed to the knob constructor. Finally, developers can optionally define a *knob constraint*: an arbitrary Boolean expression over the knobs. GOAL's runtime then filters out any configurations that violate these knob constraints before passing the model to the AdaptLog.

6 Evaluation Methodology

We evaluate the following:

- **Generality**: Can GOAL support a wide range of metrics and knobs while meeting goals as well as prior approaches designed for specific goals and knobs?
- **Dynamism**: When meta-adaptation is performed, does GOAL converge to the new AdaptSpec while providing near-optimal behavior for the objective function?
- **Robustness**: Does GOAL reliably meet goals even in the face of multi-module adaptation, errors in the learned model, and time varying workloads?

This section details the applications, platforms, points of comparison, and metrics used to evaluate these properties.

6.1 Applications and Platforms Evaluated

We implement 7 adaptive applications from prior work and then augment them to perform meta-adaptation. We start

Table 3. Properties of applications used to evaluate GOAL.

Name	Platform	# App. Knobs	# Sys. Knobs	Initial Objective	Initial Constraint	Change after Meta-adaptation
CCTV Camera [41]	Embedded	3	2	max(quality)	thruput == 30	min(power)
Video Object Detector [63]	Embedded	4	2	min(power)	thruput == 20	Restrict QP
Service Oriented Architecture (SOA) [27]	Server	3	N/A	max(reliability)	latency == 0.5	min(cost), rel == 0.6
Synthetic Aperture Radar (SAR) [82]	Embedded	4	2	max(quality)	thruput == 80	max(thruput), quality == 0.7
AES Encryption [34]	Embedded	1	2	max(thruput)	power == 1.5	max(thruput/power),blksize == 256
Search Engine [6, 44]	Server	1	2	min(power)	thruput == 18.0	Restrict searched doc.
Optical Character Recognition (OCR) [2]	Server	1	2	min(power)	thruput == 8.0	max(quality), thruput == 8.0.

each with an AdaptSpec equivalent to one used in existing literature. At runtime, however, we change the goals or knobs. Table 3 shows the details, including the number of application and system knobs, the initial objective and constraint, and the required meta-adaptation. The applications cover a wide range of system and application knobs, metrics, and goals; and they exhibit nonlinear knob interactions.

To further demonstrate generality, we use two system platforms with distinct knobs. Four applications target an embedded system: an ARMv7 based ODROID-XU3 (Exynos5422) with 2GB of RAM, running Ubuntu 16.04 (GNU/ Linux 3.10). Three target a server: an x86 (Intel i7-6700) with 8GB of RAM, running Ubuntu 18.04 (GNU/Linux 5.30). All use multithreading equal to the core count.

6.2 Adaptation Approaches Compared

We compare GOAL to prior approaches including:

- **Linear Control**: This approach uses a single linear control model for all applications [40].
- **Application-specific AdaptLogs**: Using prior work, we synthesize a specialized AdaptLog for each application's specific AdaptSpec [26]. This approach can meet each application's initial AdaptSpec, but not the new one created through meta-adaptation.
- **Multiple Application-specific AdaptLogs**: We synthesize a specialized AdaptLog for each application's AdaptSpecs both before and after meta-adaptation. At runtime, we perform meta-adaptation by shutting down the first AdaptLog and initializing the second. This approach is clearly impractical as users must know the new goals and knobs ahead of time.
- **Oracle:** We exhaustively search the configuration space to find the best knob configuration for all AdaptSpecs.

6.3 Evaluation Metrics

We quantify the approaches using the following metrics:

- **Mean Absolute Percentage Error (MAPE)**: Each application's AdaptSpec has a constraint for a particular metric. To calculate MAPE, at each call to `optimize()` we measure the absolute error between the constraint and the achieved metric, then take the mean over all iterations. Lower is better.

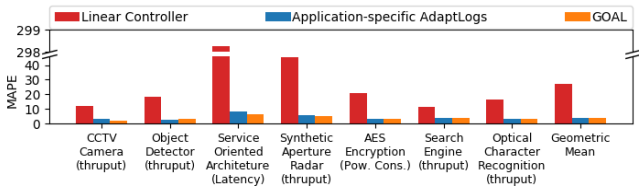

Figure 6. GOAL reliably meets the constraint of adaptation.

- **Normalized Performance**: The AdaptSpecs also specify an objective, or a metric to be optimized. To measure how well GOAL optimizes the objectives we normalize to the Oracle's performance. Higher is better as the oracle will achieve an optimum performance of 1.
- **Iterations until Convergence**: We measure the iterations required to converge to the goal after meta-adaptation. This metric will be used to compare GOAL to the approach that pre-synthesizes multiple, application specific AdaptLogs. Lower is better as it represents faster convergence after meta-adaptation.

7 Evaluation And Observations

This section evaluates key aspects of GOAL, specifically:

1. Does GOAL meet user-defined constraints across a range of knobs and metrics?
2. Does GOAL achieve near-optimal objectives when performing meta-adaptation?
3. Does GOAL converge quickly after meta-adaptation?
4. How much user effort does GOAL require?
5. Does GOAL support multi-module adaptation?
6. Is GOAL robust to errors in modeling?
7. Is GOAL robust to large changes in workload?
8. How much overhead does GOAL incur?

7.1 Does GOAL meet user constraints?

For each, we measure the MAPE for the application-specified constraints (as listed in Table 3). Figure 6 shows the results with MAPE on the vertical axis, applications on the horizontal axis, and a bar for each AdaptLog. Due to the complicated and varied application behavior, the Linear Controller fails to reliably meet the constraint exhibiting a mean MAPE of ~29%. However, both GOAL and Application-specific AdaptLogs meet the goal reliably, exhibiting less that 4% MAPE. We note that the results for the Application-specific AdaptLogs and Multiple Application-specific AdaptLogs are the same for

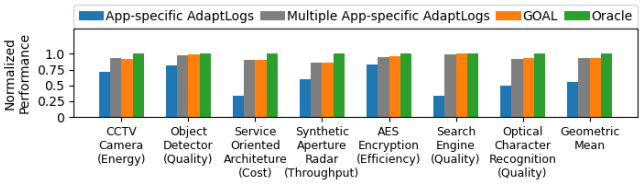

Figure 7. GOAL is optimal when meta-adaptation is needed.

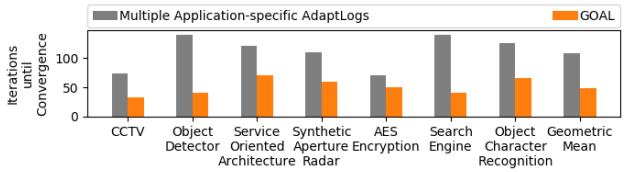

Figure 8. GOAL's runtime allows rapid convergence after meta-adaptation.

this experiment, so we omit the latter for clarity. *In summary, GOAL provides a single AdaptLog implementation that generalizes across a range of metrics and knobs while achieving errors comparable to approaches that do not generalize.*

7.2 Does GOAL optimize objectives?

We now consider the normalized performance when each application performs meta-adaptation, changing its AdaptSpecs as listed in Table 3. We report results for application-specific AdaptLogs, Multiple application-specific AdaptLogs, GOAL, and the Oracle. We omit the Linear Controller results because it fails to meet the constraints (as shown in the previous section), so its performance is not meaningful.

Figure 7 shows the results. The single Application-specific AdaptLog has poor performance because it must use the initial AdaptSpec throughout execution. However, the versions using Multiple application-specific AdaptLogs and GOAL improve on relevant metrics by 1.69× over application specific AdaptLogs, while achieving ~93.7% of the Oracle's performance.

These results show that when an application's requirements change, the application performs suboptimally with prior work that does not support meta-adaptation. GOAL, however, performs as well as an approach that knows how the requirements will change ahead of time and synthesizes multiple application-specific AdaptLogs for both requirements. *This demonstrates GOAL's support for dynamic changes in adaptation specifications, as it performs near optimally when applications trigger meta-adaptation.*

7.3 How quickly does GOAL converge?

The previous section shows that GOAL's average performance is equivalent to an approach that is custom built for a specific AdaptSpec. We now show that GOAL provides an additional benefit by measuring how quickly each approach (Multiple Application-specific AdaptLogs and GOAL) reconverges to the new constraint after meta-adaptation.

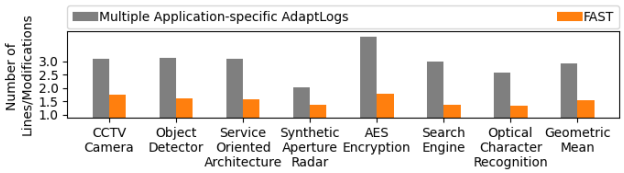

Figure 9. GOAL requires minimal development effort for adding adaptation and meta-adaptation.

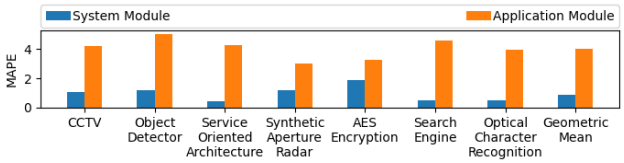

Figure 10. GOAL reliably meets goals of multiple modules.

Figure 8 shows the results with iterations required for convergence on the vertical axis. GOAL converges ~2.14× faster than using multiple application-specific AdaptLogs. The application-specific approach takes longer due to the expense of destroying the old AdaptLog and initializing the new one. This process incurs two sources of overhead: (1) the time required to perform these costly operations and (2) a loss of accumulated history of observed metric values, which must be collected by the new AdaptLog from scratch. Such history is crucial for any control or learning based AdaptLog and without it, convergence is delayed. This ability to retain history across multiple metrics and knobs is an advantage to a single, general adaptation framework like GOAL. GOAL converges quicker to the new goals because the runtime tracks all relevant metrics at all times, ensuring that no data is lost when performing meta-adaptation. *These results support the claim of dynamism by showing that GOAL converges much quicker than combining existing AdaptLogs without explicit support for meta-adaptation.*

7.4 How much effort does GOAL require?

We count the lines of code that need to be added/modified to the original non-adaptive versions of the applications. We do not count lines of code to implement the AdaptLog. Here we only compare GOAL and Multiple Application-specific AdaptLogs because these are the only two approaches which can implement meta-adaptation.

Figure 9 shows the number of lines added/modified on the vertical axis. Multiple-application specific AdaptLogs require around 1.92× more changes than GOAL. *These results show that GOAL's interface facilitates adding adaptation in a wide range of applications without significant development effort, and much less than trying to dynamically switch between prior approaches.* Finally, note that GOAL can support any type of meta-adaptation, while prior work takes more effort and yet only supports the AdaptSpecs specific to each application.

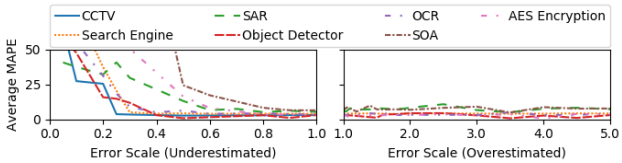

Figure 11. GOAL is robust to substantial error in profiling.

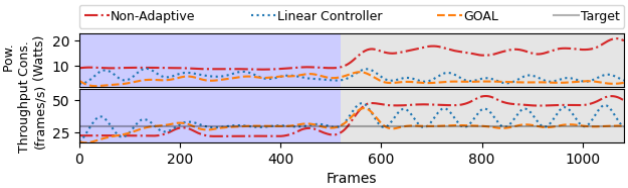

Figure 12. GOAL is robust to shifts in workload.

7.5 Does GOAL support multiple modules?

Results so far have used a single `optimize()` call per application. However, in many scenarios, multiple developers may each want to contribute a module with its own `optimize()` method (as described in § 5.2). Here, we evaluate how well GOAL performs in such multi-module adaptation scenarios.

Specifically, for each application, we implement two modules: one for the application knobs and one for system knobs. The application modules meet the constraints from Table 3. The system modules meet a power constraint while delivering maximum performance. The challenge here is that both modules affect performance, meaning that even though the knobs are independent they still affect each other.

We then measure the MAPE for both the application and system modules. Figure 10 shows the results. Both the application and system modules meet their constraints reliably, exhibiting a low mean error of ~4%. *These results show that GOAL effectively deals with complex, multi-module adaptation, a capability that other approaches do not support.*

7.6 Is GOAL robust to errors in modeling?

As mentioned in § 4.3, GOAL development includes modeling the application. We now evaluate how sensitive GOAL is to possible errors in the model learning step. For each application, we introduce errors by scaling all the model's estimates (i.e., the $\hat{xup}_k$ values used in Equations 8–10) by an error factor and perform the same experiments as above.

Figure 11 shows two charts with MAPE (vertical axis) for each application given a model that is scaled by the value on the horizontal axis; the left chart shows under-estimates, the right chart shows over-estimates. For underestimated models, GOAL performs reliably in the presence of large errors, producing a MAPE that stays relatively low at a value that is roughly equivalent to results from above (Figure 6) until scaled down to 0.55. Thus, the model has to be significantly underestimated before GOAL fails to meet its constraints. In contrast, MAPE is not impacted by overestimated models. This suggests that GOAL can support a wide range of learners for model building, but they should be biased slightly towards overestimates. *Overall, these results illustrate that GOAL is robust to substantial errors in modeling.*

7.7 Is GOAL robust to changes in workload?

Adaptive systems, in general, should meet goals despite unpredictable, external disturbances. We now demonstrate that GOAL provides this capability by adapting to different scenes for a video encoder. The first is a high-motion soccer game and the second is a low-motion news cast. Without adaptation—i.e., with a constant knob configuration—the scene change causes significant changes in throughput because different levels of motion require significantly different amounts of work to encode.

Figure 12 shows the power and throughput for three different video encoder implementations: (1) Non-Adaptive, using the fixed, default knob configuration; (2) Linear Controller, similar to the one above, but now we carefully calibrate the control to meet the goal for the initial scene, and (3) GOAL. The different colored regions show two different input scenes. The impact of scene changes can be seen in the execution of Non-adaptive whose throughput and power change from scene to scene. Linear Controller and GOAL both have a throughput constraint of 30.0 frames/s. The changing workload negatively impacts Linear Controller, which, is stable only for the first scene, but then starts to oscillate after the scene change and overall has a high MAPE of 22.5%. However, GOAL copes with these changes and meets the goal reliably during all scenes, exhibiting a much lower MAPE of 2.8%. The Linear Controller fails because the difference in the base throughput (m_{base} from Equation 5) is over a factor of 2× different from scene to scene. GOAL can handle this change, however, because it continually estimates the time-varying base behavior to account for these types of changes. *These results further demonstrate the value of GOAL's adaptive control approach that adjusts the control to handle non-linearities in application workload.(§ 4.1.2).*

7.8 How much overhead does GOAL incur?

We evaluate the overhead of performing adaptation and meta-adaptation in terms of the geometric mean of times required to perform essential operations using GOAL.

Adaptation only requires computing schedules using the AdaptLog (Equations 4–10). The time to compute a single schedule in the embedded applications is ~0.68ms, while for server applications it is ~0.045ms. Obviously, these numbers are highly influenced by the underlying hardware. Schedules are only computed once per window (§ 4.1.7). Hence, the total overhead per iteration becomes negligible.

Meta-adaptation requires calls to `intend`, `restrict` and `control`. The time required for `intend` in the embedded applications is ~5.14ms, while in the server applications it is ~1.07ms. The times for calls to `restrict` and `control` in

the embedded applications is ~11.79ms and ~8.31ms, respectively. These times in the server applications are ~0.68ms and ~0.55ms, respectively. This overhead includes the time to recompile the AdaptSpec and compute the new lookup table (§ 4.1.7). GOAL adjusts the goal for the window immediately after meta-adaptation is performed. For all subsequent windows, the goal is set to the value in the AdaptSpecs. This means that, GOAL accounts for its own effect on metrics and ensures constraints are met despite GOAL's own overhead. It should be noted that applications are expected to perform meta-adaptation far less frequently than they compute schedules. *These results show that GOAL incurs low overhead for both adaptation and meta-adaptation.*

8 Conclusion

This work motivates the benefits of supporting general purpose adaptation and meta-adaptation. We implement this idea as GOAL, a first-of-its-kind adaptation framework which, unlike prior work, is not restricted to a particular set of knobs and goals. Instead GOAL uses a virtualized AdaptLog that is parameterized by a model after the developers have declared application knobs and metrics. This allows GOAL applications to define the AdaptSpecs, using wide range of knobs and metrics, and seamlessly modify them during execution. The AdaptLog is, itself, adaptive and therefore robust to adapt to nonlinear behavior and changing workloads and operating conditions. We show that GOAL's general approach handles a diverse range of goals and knobs, while performing just as well as problem specific AdaptLogs. However, when the application requires meta-adaptation, GOAL performs significantly better than prior work. Furthermore, GOAL provides an easy-to-use interface that requires minimal effort to add adaptation and meta-adaptation to applications and even supports applications with multiple, independent adaptive modules. We believe that GOAL's generality, dynamism and robustness goes a long way to fulfilling the requirements of complex emerging systems.

9 Acknowledgments

This work has been supported by NSF (grants CCF-2119184, CNS-1956180, CNS-1952050, CCF-1823032, CNS-1764039), ARO (grant W911NF1920321), a DOE Early Career Award (grant DESC0014195 0003), and the Proteus project under the DARPA BRASS program.

References

[1] [n. d.]. Arlo: Wire-Free HD and HDR Smart Home Security Cameras. https://www.arlo.com/en-us/default.aspx

[2] [n. d.]. Tesseract OCR. https://opensource.google/projects/tesseract

[3] Frederico Alvares, Gwenaël Delaval, Eric Rutten, and Lionel Seinturier. 2017. Language Support for Modular Autonomic Managers in Reconfigurable Software Components. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. 271–278. https://doi.org/10.1109/ICAC.2017.48

[4] Jason Ansel, Maciej Pacula, Yee Lok Wong, Cy Chan, Marek Olszewski, Una-May O'Reilly, and Saman Amarasinghe. 2012. Siblingrivalry: Online Autotuning Through Local Competitions. In *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems* (Tampere, Finland) *(CASES '12)*. ACM, New York, NY, USA, 91–100. https://doi.org/10.1145/2380403.2380425

[5] Apple. 2019. Swift. https://developer.apple.com/swift/.

[6] Woongki Baek and Trishul M. Chilimbi. 2010. Green: A Framework for Supporting Energy-conscious Programming Using Controlled Approximation. In *Proceedings of the 31st ACM Conference on Programming Language Design and Implementation* (Toronto, Ontario, Canada) *(PLDI '10)*. ACM, New York, NY, USA, 198–209. https://doi.org/10.1145/1806596.1806620

[7] Baochun Li and K. Nahrstedt. 1999. A control-based middleware framework for quality-of-service adaptations. *IEEE Journal on Selected Areas in Communications* 17, 9 (1999), 1632–1650. https://doi.org/10.1109/49.790486

[8] Saeid Barati, Ferenc A. Bartha, Swarnendu Biswas, Robert Cartwright, Adam Duracz, Donald Fussell, Henry Hoffmann, Connor Imes, Jason Miller, Nikita Mishra, Arvind, Dung Nguyen, Krishna V. Palem, Yan Pei, Keshav Pingali, Ryuichi Sai, Andrew Wright, Yao-Hsiang Yang, and Sizhuo Zhang. 2019. Proteus: Language and Runtime Support for Self-Adaptive Software Development. *IEEE Software* 36, 2 (2019), 73–82. https://doi.org/10.1109/MS.2018.2884864

[9] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. 2010. Towards Energy-aware Scheduling in Data Centers Using Machine Learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking* (Passau, Germany) *(e-Energy '10)*. ACM, New York, NY, USA, 215–224. https://doi.org/10.1145/1791314.1791349

[10] James Bornholt, Todd Mytkowicz, and Kathryn S. McKinley. 2014. Uncertain<T>: A First-Order Type for Uncertain Data. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 51–66. https://doi.org/10.1145/2541940.2541958

[11] John Boyd. 1995. The Essence of Winning and Losing. Online document. https://www.danford.net/boyd/essence.htm

[12] John Raymond Boyd. 1976. Destruction and Creation.

[13] John Raymond Boyd. 1996. The Essence of Winning and Losing.

[14] Stephen P. Bradley, Arnoldo C. Hax, and Thomas L. Magnanti. 1977. *Applied mathematical programming*. Addison-Wesley.

[15] Anthony Canino and Yu David Liu. 2017. Proactive and Adaptive Energy-aware Programming with Mixed Typechecking. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. ACM, New York, NY, USA, 217–232. https://doi.org/10.1145/3062341.3062356

[16] Anthony Canino, Yu David Liu, and Hidehiko Masuhara. 2018. Stochastic Energy Optimization for Mobile GPS Applications. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. ACM, New York, NY, USA, 703–713. https://doi.org/10.1145/3236024.3236076

[17] Liyu Cao and Howard M. Schwartz. 2004. Analysis of the Kalman filter based estimation algorithm: an orthogonal decomposition approach. *Automatica* 40, 1 (2004), 5–19. https://doi.org/10.1016/j.automatica.2003.07.011

[18] Jacques Carette, Oleg Kiselyov, and Chung-chieh Shan. 2009. Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages. *Journal of Functional Programming* 19, 5 (2009), 509–543.

[19] H. Chen, M. Song, J. Song, A. Gavrilovska, and K. Schwan. 2011. HEaRS: A Hierarchical Energy-Aware Resource Scheduler for Virtualized Data Centers. In *2011 IEEE International Conference on Cluster Computing*. 508–512. https://doi.org/10.1109/CLUSTER.2011.60

[20] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. *SIGPLAN Not.* 48, 4 (March 2013), 77–88. https://doi.org/10.1145/2499368.2451125

[21] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-Aware Cluster Management. *SIGPLAN Not.* 49, 4 (Feb. 2014), 127–144. https://doi.org/10.1145/2644865.2541941

[22] Yi Ding, Nikita Mishra, and Henry Hoffmann. 2019. Generative and Multi-Phase Learning for Computer Systems Optimization. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 39–52. https://doi.org/10.1145/3307650.3326633

[23] Yi Ding, Nikita Mishra, and Henry Hoffmann. 2019. Generative and Multi-Phase Learning for Computer Systems Optimization. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. Association for Computing Machinery, New York, NY, USA, 39–52. https://doi.org/10.1145/3307650.3326633

[24] Mirko D'Angelo. 2018. Decentralized Self-Adaptive Computing at the Edge. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems* (Gothenburg, Sweden) *(SEAMS '18)*. Association for Computing Machinery, New York, NY, USA, 144–148. https://doi.org/10.1145/3194133.3194160

[25] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Architecture Support for Disciplined Approximate Programming. *SIGARCH Comput. Archit. News* 40, 1 (March 2012), 301–312. https://doi.org/10.1145/2189750.2151008

[26] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2014. Automated Design of Self-adaptive Software with Control-theoretical Formal Guarantees. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. ACM, New York, NY, USA, 299–310. https://doi.org/10.1145/2568225.2568272

[27] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2015. Automated Multi-objective Control for Self-adaptive Software Design. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. ACM, New York, NY, USA, 13–24. https://doi.org/10.1145/2786805.2786833

[28] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro Vittorio Papadopoulos, Suprio Ray, Amir Molzam Sharifloo, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. 2017. Control Strategies for Self-Adaptive Software Systems. *TAAS* 11, 4 (2017). https://doi.org/10.1145/3024188

[29] Jason Flinn and M. Satyanarayanan. 1999. Energy-Aware Adaptation for Mobile Applications. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles* (Charleston, South Carolina, USA) *(SOSP '99)*. Association for Computing Machinery, New York, NY, USA, 48–63. https://doi.org/10.1145/319151.319155

[30] Andrei Frumusanu. 2018. Improving The Exynos 9810 Galaxy S9: Part 1. https://www.anandtech.com/show/12615/improving-exynos-9810-galaxy-s9-part-1.

[31] Andrei Frumusanu. 2018. The Samsung Galaxy S9 and S9+ Review: Exynos and Snapdragon at 960fps. https://www.anandtech.com/show/12520/the-galaxy-s9-review/5.

[32] Jim Gao. 2014. Machine Learning Applications for Data Center Optimization.

[33] Ashvin Goel, David Steere, Calton Pu, and Jonathan Walpole. 1998. *SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit.* Technical Report.

[34] J. Goodman, A. P. Dancy, and A. P. Chandrakasan. 1998. An energy/security scalable encryption processor using an embedded variable voltage DC/DC converter. *IEEE Journal of Solid-State Circuits* 33, 11 (Nov 1998), 1799–1809. https://doi.org/10.1109/4.726580

[35] Google. 2020. Android Power Management: Battery Saver. https://developer.android.com/about/versions/pie/power#battery-saver.

[36] Guang-Liang Li. 1995. An analysis of impact of workload fluctuations on performance of computer systems. In *Proceedings of 1995 IEEE International Computer Performance and Dependability Symposium*. 256–264. https://doi.org/10.1109/IPDS.1995.395826

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

[38] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S. Gunawi. 2020. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 173–190. https://www.usenix.org/conference/osdi20/presentation/hao

[39] Joseph L. Hellerstein. 2009. Engineering Autonomic Systems. In *Proceedings of the 6th International Conference on Autonomic Computing* (Barcelona, Spain) *(ICAC '09)*. Association for Computing Machinery, New York, NY, USA, 75–76. https://doi.org/10.1145/1555228.1555254

[40] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. 2004. *Feedback Control of Computing Systems.* John Wiley & Sons.

[41] Henry Hoffmann. 2015. JouleGuard: Energy Guarantees for Approximate Applications. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. ACM, New York, NY, USA, 198–214. https://doi.org/10.1145/2815400.2815403

[42] Henry Hoffmann, Jim Holt, George Kurian, Eric Lau, Martina Maggio, Jason E. Miller, Sabrina M. Neuman, Mahmut Sinangil, Yildiz Sinangil, Anant Agarwal, Anantha P. Chandrakasan, and Srinivas Devadas. 2012. Self-Aware Computing in the Angstrom Processor. In *Proceedings of the 49th Annual Design Automation Conference* (San Francisco, California) *(DAC '12)*. Association for Computing Machinery, New York, NY, USA, 259–264. https://doi.org/10.1145/2228360.2228409

[43] Henry Hoffmann, Jim Holt, George Kurian, Eric Lau, Martina Maggio, Jason E. Miller, Sabrina M. Neuman, Mahmut Sinangil, Yildiz Sinangil, Anant Agarwal, Anantha P. Chandrakasan, and Srinivas Devadas. 2012. Self-Aware Computing in the Angstrom Processor. In *Proceedings of the 49th Annual Design Automation Conference* (San Francisco, California) *(DAC '12)*. Association for Computing Machinery, New York, NY, USA, 259–264. https://doi.org/10.1145/2228360.2228409

[44] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic Knobs for Responsive Power-Aware Computing. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Newport Beach, California, USA) *(ASPLOS XVI)*. Association for Computing Machinery, New York, NY, USA, 199–212. https://doi.org/10.1145/1950365.1950390

[45] Petr Jan Horn. 2001. Autonomic Computing: IBM's Perspective on the State of Information Technology.

[46] Y. Hsu, K. Matsuda, and M. Matsuoka. 2018. Self-Aware Workload Forecasting in Data Center Power Prediction. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 321–330.

[47] Jian Huang, Xuechen Zhang, and Karsten Schwan. 2015. Understanding issue correlations: a case study of the Hadoop system. In *SoCC*.

[48] Connor Imes, Lars Bergstrom, and Henry Hoffmann. 2016. A Portable Interface for Runtime Energy Monitoring. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 968–974. https://doi.org/10.1145/2950290.2983956

[49] Connor Imes and Henry Hoffmann. 2016. Bard: A unified framework for managing soft timing and power constraints. In *Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016 International Conference on*. IEEE, 31–38.

[50] Connor Imes, Steven Hofmeyr, and Henry Hoffmann. 2018. Energy-Efficient Application Resource Scheduling Using Machine Learning Classifiers. In *Proceedings of the 47th International Conference on Parallel Processing* (Eugene, OR, USA) *(ICPP 2018)*. Association for Computing Machinery, New York, NY, USA, Article 45, 11 pages. https://doi.org/10.1145/3225058.3225088

[51] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. 2015. POET: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*. 75–86. https://doi.org/10.1109/RTAS.2015.7108419

[52] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2009. Self-Adaptive and Self-Configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters. In *Proceedings of the 6th International Conference on Autonomic Computing* (Barcelona, Spain) *(ICAC '09)*. Association for Computing Machinery, New York, NY, USA, 117–126. https://doi.org/10.1145/1555228.1555261

[53] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2014. Adaptive Resource Provisioning for Virtualized Servers Using Kalman Filters. *ACM Trans. Auton. Adapt. Syst.* 9, 2, Article 10 (July 2014), 35 pages. https://doi.org/10.1145/2626290

[54] Aman Kansal, Scott Saponas, A.J. Bernheim Brush, Kathryn S. McKinley, Todd Mytkowicz, and Ryder Ziola. 2013. The Latency, Accuracy, and Battery (LAB) Abstraction: Programmer Productivity and Energy Efficiency for Continuous Mobile Context Sensing. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications* (Indianapolis, Indiana, USA) *(OOPSLA '13)*. ACM, New York, NY, USA, 661–676. https://doi.org/10.1145/2509136.2509541

[55] J. O. Kephart and D. M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (Jan 2003), 41–50. https://doi.org/10.1109/MC.2003.1160055

[56] Minyoung Kim, Mark-Oliver Stehr, Carolyn Talcott, Nikil Dutt, and Nalini Venkatasubramanian. 2013. XTune: A Formal Methodology for Cross-Layer Tuning of Mobile Embedded Systems. *ACM Trans. Embed. Comput. Syst.* 11, 4, Article 73 (Jan. 2013), 23 pages. https://doi.org/10.1145/2362336.2362340

[57] Karl Krauth, Stephen Tu, and Benjamin Recht. 2019. Finite-time Analysis of Approximate Policy Iteration for the Linear Quadratic Regulator. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8514–8524. http://papers.nips.cc/paper/9058-finite-time-analysis-of-approximate-policy-iteration-for-the-linear-quadratic-regulator.pdf

[58] Robert Laddaga. 1999. Guest Editor's Introduction: Creating Robust Software Through Self-Adaptation. *IEEE Intelligent Systems* 14, 3 (May 1999), 26–29. https://doi.org/10.1109/MIS.1999.769879

[59] Guang-Liang Li and P. Dowd. 1995. An analysis of network performance degradation induced by workload fluctuations. *IEEE/ACM Transactions on Networking* 3, 04 (jul 1995), 433–440. https://doi.org/10.1109/90.413217

[60] Kiwan Maeng and Brandon Lucia. 2018. Adaptive Dynamic Checkpointing for Safe Efficient Intermittent Computing. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 129–144. https://www.usenix.org/conference/osdi18/presentation/maeng

[61] Martina Maggio, Henry Hoffmann, Alessandro V. Papadopoulos, Jacopo Panerati, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva. 2012. Comparison of Decision-Making Strategies for Self-Optimization in Autonomic Computing Systems. *ACM Trans. Auton. Adapt. Syst.* 7, 4, Article 36 (Dec. 2012), 32 pages. https://doi.org/10.1145/2382570.2382572

[62] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Automated Control of Multiple Software Goals Using Multiple Actuators. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 373–384. https://doi.org/10.1145/3106237.3106247

[63] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann. 2017. Self-Adaptive Video Encoder: Comparison of Multiple Adaptation Strategies Made Simple. In *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 123–128. https://doi.org/10.1109/SEAMS.2017.16

[64] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 197–210. https://doi.org/10.1145/3098822.3098843

[65] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. 2018. CALOREE: Learning Control for Predictable Latency and Low Energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA, USA) *(ASPLOS '18)*. Association for Computing Machinery, New York, NY, USA, 184–198. https://doi.org/10.1145/3173162.3173184

[66] A. Moustafa and M. Zhang. 2014. Learning Efficient Compositions for QoS-Aware Service Provisioning. In *2014 IEEE International Conference on Web Services*. 185–192. https://doi.org/10.1109/ICWS.2014.37

[67] Frans P. B. Osinga. 2007. *Science, Strategy and War: The strategic theory of John Boyd*. Routledge.

[68] A. Padovitz, S. Loke, and A. Zaslavsky. 2003. Awareness and Agility for Autonomic Distributed Systems: Platform-Independent Publish-Subscribe Event-Based Communication for Mobile Agents. In *2012 23rd International Workshop on Database and Expert Systems Applications*. IEEE Computer Society, Los Alamitos, CA, USA, 669. https://doi.org/10.1109/DEXA.2003.1232098

[69] Ariel Rabkin and Randy Howard Katz. 2013. How hadoop clusters break. *IEEE software* 30, 4 (2013).

[70] Amir M. Rahmani, Bryan Donyanavard, Tiago Mück, Kasra Moazzemi, Axel Jantsch, Onur Mutlu, and Nikil Dutt. 2018. SPECTR: Formal Supervisory Control and Coordination for Many-Core Systems Resource Management. *SIGPLAN Not.* 53, 2 (March 2018), 169–183. https://doi.org/10.1145/3296957.3173199

[71] Redhat. 2020. Tuned: Tuning Profile Delivery Mechanism for Linux. https://tuned-project.org/.

[72] Eric Rutten, Nicolas Marchand, and Daniel Simon. 2017. Feedback Control as MAPE-K Loop in Autonomic Computing. In *Software Engineering for Self-Adaptive Systems III. Assurances*, Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese (Eds.). Springer International Publishing, Cham, 349–373.

[73] M. Révay and M. Líška. 2017. OODA loop in command control systems. In *2017 Communication and Information Technologies (KIT)*. 1–4. https://doi.org/10.23919/KIT.2017.8109463

[74] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate Data Types for Safe and General Low-power Computation. In *Proceedings of the 32Nd ACM Conference on Programming Language Design and Implementation* (San Jose, California, USA) *(PLDI '11)*. ACM, New York, NY, USA, 164–174. https://doi.org/10.1145/1993498.1993518

[75] Muhammad Husni Santriaji and Henry Hoffmann. 2016. GRAPE: Minimizing energy for GPU applications with performance requirements. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–13. https://doi.org/10.1109/MICRO.2016.7783719

[76] Homeland Security. 2013. CCTV Technology Handbook. Online Documen.

[77] Akbar Sharifi, Shekhar Srikantaiah, Asit K. Mishra, Mahmut Kandemir, and Chita R. Das. 2011. METE: Meeting End-to-End QoS in Multicores through System-Wide Resource Management. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and*

Modeling of Computer Systems (San Jose, California, USA) *(SIGMETRICS '11)*. Association for Computing Machinery, New York, NY, USA, 13–24. https://doi.org/10.1145/1993744.1993747

[78] 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* (Nuremberg, Germany) *(EuroSys '09)*. Association for Computing Machinery, New York, NY, USA, 289–302. https://doi.org/10.1145/1519065.1519097

[79] Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. 2007. Eon: A Language and Runtime System for Perpetual Systems. In *In Proceedings of The Fifth International ACM Conference on Embedded Networked Sensor Systems (SenSys '07), Syndey*.

[80] Akshitha Sriraman and Thomas F. Wenisch. 2018. µTune: Auto-Tuned Threading for OLDI Microservices. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 177–194. https://www.usenix.org/conference/osdi18/presentation/sriraman

[81] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. 1999. A Feedback-Driven Proportion Allocator for Real-Rate Scheduling. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (New Orleans, Louisiana, USA) *(OSDI '99)*. USENIX Association, USA, 145–158.

[82] Xin Sui, Andrew Lenharth, Donald S. Fussell, and Keshav Pingali. 2016. Proactive Control of Approximate Programs. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, Atlanta, GA, USA, April 2-6, 2016*. 607–621. https://doi.org/10.1145/2872362.2872402

[83] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, Ben Christensen, Alex Gartrell, Maxim Khutornenko, Sachin Kulkarni, Marcin Pawlowski, Tuomas Pelkonen, Andre Rodrigues, Rounak Tibrewal, Vaishnavi Venkatesan, and Peter Zhang. 2020. Twine: A Unified Cluster Management System for Shared Infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 787–803. https://www.usenix.org/conference/osdi20/presentation/tang

[84] Konstantinos Tovletoglou, Lev Mukhanov, Dimitrios S. Nikolopoulos, and Georgios Karakonstantis. 2020. HaRMony: Heterogeneous-Reliability Memory and QoS-Aware Energy Management on Virtualized Servers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 575–590. https://doi.org/10.1145/3373376.3378489

[85] Stephen Tu and Benjamin Recht. 2018. Least-Squares Temporal Difference Learning for the Linear Quadratic Regulator. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 5005–5014. http://proceedings.mlr.press/v80/tu18a.html

[86] David Vengerov. 2009. A reinforcement learning framework for utility-based scheduling in resource-constrained systems. *Future Generation Computer Systems* 25, 7 (2009), 728 – 736. https://doi.org/10.1016/j.future.2008.02.006

[87] Jóakim von Kistowski, Hansfried Block, John Beckett, Klaus-Dieter Lange, Jeremy Arnold, and Samuel Kounev. 2015. Analysis of the Influences on Server Power Consumption and Energy Efficiency for CPU-Intensive Workloads. *ICPE 2015 - Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. https://doi.org/10.1145/2668930.2688057

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

[89] Hongbign Wang, Xin Chen, Qin Wu, Qi Yu, Xingguo Hu, Zibin Zheng, and Athman Bouguettaya. 2017. Integrating Reinforcement Learning with Multi-Agent Techniques for Adaptive Service Composition. *ACM Trans. Auton. Adapt. Syst.* 12, 2, Article 8 (May 2017), 42 pages. https://doi.org/10.1145/3058592

[90] Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. 2018. Understanding and Auto-Adjusting Performance-Sensitive Configurations. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA, USA) *(ASPLOS '18)*. ACM, New York, NY, USA, 154–168. https://doi.org/10.1145/3173162.3173206

[91] Yanzhi Wang and Massoud Pedram. 2016. Model-Free Reinforcement Learning and Bayesian Classification in System-Level Power Management. *IEEE Trans. Comput.* 65, 12 (Dec 2016), 3713–3726. https://doi.org/10.1109/TC.2016.2543219

[92] Greg Welch and Gary Bishop. 1995. *An Introduction to the Kalman Filter*. Technical Report. USA.

[93] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. 2018. Neural Adaptive Content-aware Internet Video Delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 645–661. https://www.usenix.org/conference/osdi18/presentation/yeo

[94] Wanghong Yuan and Klara Nahrstedt. 2003. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA) *(SOSP '03)*. Association for Computing Machinery, New York, NY, USA, 149–163. https://doi.org/10.1145/945445.945460

[95] Ronghua Zhang, Chenyang Lu, Tarek F. Abdelzaher, and John A. Stankovic. 2002. ControlWare: a middleware architecture for feedback control of software performance. In *Proceedings 22nd International Conference on Distributed Computing Systems*. 301–310. https://doi.org/10.1109/ICDCS.2002.1022267

[96] Yuhao Zhu and Vijay Janapa Reddi. 2016. GreenWeb: Language Extensions for Energy-efficient Mobile Web Computing. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Santa Barbara, CA, USA) *(PLDI '16)*. ACM, New York, NY, USA, 145–160. https://doi.org/10.1145/2908080.2908082

Received 2022-07-12; accepted 2022-10-02