Combining Machine Learning and Control to Manage Computing System Complexity and Dynamics*

Nikita Mishra Connor Imes
University of Chicago
nmishra,ckimes@cs.uchicago.edu

John D. Lafferty
Yale University
john.lafferty@yale.edu

Henry Hoffmann
University of Chicago
hankhoffmann@cs.uchicago.edu

Introduction

As power and energy become first-order concerns for computing systems, software developers are increasingly tasked with meeting multiple, often conflicting goals; *e.g.,* creating responsive mobile applications that maximize battery life. Developers faced with this problem must address two challenges: *complexity* and *dynamics*.

Complexity arises as computer architectures increasingly expose resources to software for management. Consider Samsung's recent release of the Galaxy S9+ smartphone. This phone came with an upgraded multicore CPU—the first "very large core" in a smartphone. A tech reporter, however, found that performance and battery life were much worse than expected because heuristics that did a good job of core and DVFS management for the S9, worked very poorly for the S9+ (click for link). The S9+ is just one example of how hardware complexity creates problems for software: a resource management heuristic that worked well for one system (the S9) was extremely poor on another (the S9+).

Dynamics arise from fluctuating workloads and varying resource availability. Thus, even if software developers find a resource configuration that works well in one scenario, there is no guarantee that it will continue to meet goals as the environment changes. A video encoder represents a simple example. During low-motion scenes, the encoder's performance requirements (keeping up with the camera) are easily met with low resource usage. In high-motion scenes, the encoder needs more resources to produce high-quality video. If software always allocates resources for the low-motion case, quality will suffer in times of high-motion. If software allocates for high-motion, energy (and thus battery life) is wasted during simpler scenes.

Prior Work on Complexity and Dynamics

To manage the complexity of modern processors, many researchers have applied statistics, machine learning, or artificial intelligence [5, 12, 15, 31, 52, 57, 58, 66, 85].[1] These learning techniques are well suited to modeling high-level application behavior—*e.g.,* performance, power, and energy—as a function of system resource allocation. These models, however, may be invalidated if the environment changes sufficiently. Even reinforcement learning techniques—which update models dynamically—are insufficient for many deployments, because they (1) provide no formal guarantees that they will deliver the desired behavior and (2) they require software developers to manually tune parameters such as learning rate [46].

To manage computing system dynamics, a number of research projects have turned to control theoretic solutions [8, 24, 25, 30, 42, 64, 69, 74, 80, 82]. Control provides formal guarantees that it will meet goals, if they are achievable. Perhaps more importantly, control formalisms permit reasoning about the precise conditions under which the goals can be achieved [24]. Unfortunately, these guarantees are based on bounding the computer system's ground-truth behavior. If models are not available—or their error cannot be bounded—then control systems cannot be applied. These restrictions make it extremely difficult to apply control theory solutions to general-purpose computing systems. A model may not be available, and even if it is, models of application performance and resource usage are often non-linear and non-convex, making them ill-suited to most control techniques. The most successful deployment of control has thus been in application-limited scenarios where models are relatively easy to build, such as managing multimedia applications [18, 19, 35, 47, 74, 80] or webservers [29, 45, 70].

Maggio et al. empirically compare a wide range of learning and control approaches to meet application latency requirements with minimal energy through active resource management [46]. Their findings are consistent with the above observations. Reinforcement learning is the best choice if no model is available, but control systems are significantly more robust given a model.

Motivation and Challenges

Intuitively, learning and control should be combined to provide formal guarantees that a computer system will meet its goals even in a general purpose environment with no prior knowledge of the application to be controlled. *Indeed, the mechanisms and implementation that make this combination work efficiently are precisely the contributions of this paper.* In a general purpose computing environment, learning can produce highly accurate models for new applications and then pass those to a control system that ensures the goals are met; this approach would combine learning's flexibility

[1]All citation numbers refer to the original paper.

**Originally Appeared as: Mishra, Connor Imes, John D. Laferty, and Henry Hofmann. CALOREE: Learning Control for Predictable Latency and Low Energy. 184–198., ASPLOS '18: 2018 Architectural Support for Programming Languages and Operating Systems, March 24–28, 2018, Williamsburg, VA* with control's formal guarantees. Realizing this combination, however, requires addressing two major challenges:

- Dividing resource management into sub-problems that suit learning and control's different strengths.
- Defining abstractions that efficiently combine sub-problem solutions, while maintaining control's formal guarantees.

CALOREE: Combining Learning and Control

We address the first challenge by adopting a *virtualized* control system, which is easily separated into learning and control tasks. Textbook controllers manage *physical* actuators (such as clockspeed or the number of allocated cores) [24]. Thus, applying control requires knowing the precise relationship between a physical actuator and the application under control—impossible in a general environment. We instead propose a *virtualized* control system. For example, to control application latency, we use *speedup* instead of absolute performance. In this way, all unpredictable external interference is viewed as a change to a *baseline* latency and the relative speedup is insensitive to these changes. Learning is well-suited to modeling speedups as a function of resource usage and finding Pareto-optimal tradeoffs in speedup and energy. Once the learner has found Pareto-optimal tradeoffs the problem is convex, piece-wise linear, and well-suited to adaptive control solutions which guarantee the required speedup even in dynamic environments. Figure 1 illustrates the intuition: processor complexity creates local optima, where control solutions can get stuck; but learning finds true optimal tradeoffs—"smoothing"—the problem, allowing control techniques to handle dynamics while providing globally optimal resource allocations.

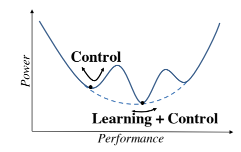

Figure 1. Learning smoothes the controller's domain.

We address the second challenge by defining a two-part interface between learning and control that maintains control's formal guarantees. The first part is a *performance hash table* (PHT) that stores the piecewise-linear learned model. The PHT allows the controller to find the resource allocation that meets a desired speedup with minimal energy and requires only constant (average case) access time. The interface's second part is the learned variance. Knowing this value, the controller automatically adjusts its internals to maintain formal guarantees even though the speedup is modeled by a noisy learning mechanism at runtime, rather than directly measured offline—as in traditional control design.

Thus, we propose a general methodology where an abstract control system is customized at runtime by a learner. We refer to this approach as CALOREE[2]. Unlike previous work on control systems that required numerous user-specified models and parameters [8, 30, 42, 64, 82], CALOREE builds models and tunes parameters automatically; *i.e., it requires no user-level inputs other than latency requirements.*

Results Summary

We evaluate CALOREE by pairing a number of different learners with our virtualized control system to manage application latency on a heterogeneous ARM big.LITTLE system. We compare to state-of-the-art learning (including polynomial regression [15, 66], collaborative filtering—*i.e.,* the Netflix algorithm[3, 12]—and a hierarchical Bayesian model [52]) and control (including proportional-integral-derivative [24] and adaptive, or self-tuning [41]) controllers. We also compare with a naive version of CALOREE that does not account for possible errors in the learner; *i.e.,* it assumes learned models are ground truth. We set latency goals for benchmark applications and measure both the percentage of time the latency requirements are violated and the energy. We test both *single-app*—where an application runs alone—and *multi-app* environments—where background applications enter the system and compete for resources.

Our paper shows that CALOREE achieves the *most reliable latency* and *best energy savings.* In the *single-app* case, the best prior technique misses 10% of deadlines on average, while CALOREE misses only 6%. All other approaches miss 100% of deadlines for at least one application, but CALOREE misses, at most, 11% of deadlines. In the *multi-app* case, the best prior approach averages 40% deadline misses, but CALOREE misses just 20% (we note that not all goals can be met in this second scenario). We evaluate energy by comparing to optimal energy assuming a perfect model of application and system. In the *single-app* case, the best prior approach averages 18% more energy consumption than optimal, but CALOREE consumes only 4% more. In the *multi-app* case, the best prior approach averages 28% more energy than optimal, while CALOREE consumes just 6% more. Critically, the naive version of CALOREE is often no better than prior approaches, showing the importance of not just constructing the models, but also incorporating possible error into the control design.

Contributions

In summary, *CALOREE is the first work to use learning to customize control systems at runtime, ensuring application latency—both formally and empirically—with no prior knowledge of the controlled application.* While the approach was originally intended to manage latency with minimal energy, the ideas are general and can be trivially extended to other goals and tradeoffs. CALOREE's contributions are:

- Separation of resource management into (1) *learning* complicated resource interactions and (2) *controlling* a virtual goal that is later mapped into specific resource settings.
- A generalized control design usable with multiple learners.
- A method for guaranteeing latency using learned—rather than measured—models.

[2]**C**ontrol **A**nd **L**earning for **O**ptimal **R**esource **E**nergy **E**fficiency

CALOREE's Potential Impact

While we originally demonstrated CALOREE on an ARM big.LITTLE architecture (running Linux) we believe CALOREE represents a general approach to resource—and even configuration—management, as all such management systems will be forced to deal with complexity and dynamics. In addition, the concept of marrying a control system to a learner has potential to make learning-based approaches much more robust and predictable. We address these two potential impacts (*general configuration management* and *enhancing learning systems*) in the remainder of this document. We note that the citation for a potential test of time award is the same as the footnote listed on the first page of this document.

Generalized Configuration Management

A resource allocation can simply be viewed as a configuration. Many software and hardware systems are configurable, but deployed systems often rely on heuristic configuration selection. These heuristics must be tuned by software developers and are extremely brittle. CALOREE provides more reliable performance (fewer missed deadlines) and better energy consumption (closer to optimal) than prior machine learning and control approaches. At the same time, CALOREE's only parameter is the desired operating point for the managed application. By eliminating user-specified parameters, CALOREE should be much more robust than heuristic-based approaches that must be tuned for each individual deployment and may have no suitable static setting.

To demonstrate this robustness, we have ported CALOREE to three new environments. First, we repeated similar experiments to the original paper on a Linux/x86 server, again finding that CALOREE provides the most reliable performance and greatest energy savings. As both the ARM system from the original paper and this x86 system run Linux, these results are achieved with the exact same code—the only changes are to configuration files specifying available resources. We then ported CALOREE to Android to test on the Galaxy S9+. The only code changes required here were Android specific resource monitoring and actuation—the math and methodology are the same. While we have only had a short time to test the S9+ we have measured web-browsing latency and found that CALOREE again provides more reliable performance with near-optimal energy while far outperforming even the best manual tuning of Samsung's scheduler.

Finally, in a DARPA collaboration with MIT, Rice, and UT Austin, we used CALOREE to dynamically configure an embedded video encoder. In this case, CALOREE configures application variants that trade performance for image quality. For the DARPA demo, a human adversary causes system (including fan and core) failures. When resources are available, CALOREE produces the highest quality video. When resources fail, CALOREE sacrifices accuracy to maintain frame rate. The same code from the original paper was used in this demo, the only difference is the specification of application-level alternatives instead of system resources. A video of the demo in action is available: https://youtu.be/3PYY6f92muY.

Enhancing Learning-based Management

CALOREE makes learning-based configuration management much more robust without requiring redesign of existing learning approaches. CALOREE works with any learner that produces predictions of application behavior as a function of system configuration. Our original paper tested four such learners and found that the combination of learning and control was always better than learning alone.

While the above is a good empirical result, there are foundational reasons that these results should hold in general. CALOREE does not simply bolt a controller to a learner, but instead automatically tunes its internal control equations to the learner's output. For example, the controller's *pole* is a key parameter. Under traditional control designs, this pole is set by the human designer to govern the control system's dynamic response and guarantee convergence to the goal. Rather than requiring a human to tune this key parameter, CALOREE automatically sets it based on the ratio of the minimum and maximum estimated speedups and the learner's confidence interval. Thus, CALOREE's methodology for combining learning and control automatically compensates for uncertainty in the learner's models.

The only requirements for the learner are that: (1) it produces a piecewise-linear model relating configurations to behavior and (2) provides confidence intervals. Under these assumptions, CALOREE formally guarantees that the application will meet its goals. The difference from traditional control is that CALOREE provides probabilistic (as a function of the confidence interval) instead of absolute guarantees. (Learners which cannot provide confidence intervals can still be used with CALOREE—and were tested in the original paper—but without these values the formal guarantees are lost). Thus, there is both a theoretical and empirical basis to believe that CALOREE's approach will improve any learning-based configuration management system.

Summary of Impact

CALOREE creates more robust and efficient computing systems through intelligent configuration management. CALOREE includes a rigorous methodology for designing and deploying computing systems that are aware of high-level goals and automatically adapt their behavior to ensure those goals are met in complex, dynamic environments. Where the current state-of-the-practice involves *ad hoc*, heuristic techniques, CALOREE addresses multi-objective optimization in a fundamental way. CALOREE is portable, formally analyzable, and easily re-purposed to address new problems as they emerge. CALOREE is more robust and efficient than either learning or control alone, adapting to meet multiple goals while requiring less programmer effort.

References

[1] 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 *CASES*.

[2] Jason Ansel, Yee Lok Wong, Cy Chan, Marek Olszewski, Alan Edelman, and Saman Amarasinghe. 2011. Language and compiler support for auto-tuning variable-accuracy algorithms. In *CGO*.

[3] R. M. Bell, Y. Koren, and C. Volinsky. 2008. *The BellKor 2008 solution to the Netflix Prize.* Technical Report. ATandT Labs.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *PACT*.

[5] Ramazan Bitirgen, Engin Ipek, and Jose F. Martinez. 2008. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *MICRO*.

[6] Giorgio C Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. 2006. *Soft Real-Time Systems: Predictability vs. Efficiency: Predictability vs. Efficiency.* Springer.

[7] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IISWC*.

[8] Jian Chen and Lizy Kurian John. 2011. Predictive coordination of multiple on-chip resources for chip multiprocessors. In *ICS*.

[9] Jian Chen, Lizy Kurian John, and Dimitris Kaseridis. 2011. Modeling Program Resource Demand Using Inherent Program Characteristics. *SIGMETRICS Perform. Eval. Rev.* 39, 1 (June 2011), 1–12.

[10] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. 2011. Pack & Cap: adaptive DVFS and thread packing under power caps. In *MICRO*.

[11] Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. 2017. *On the Sample Complexity of the Linear Quadratic Regulator.* Technical Report 1710.01688v1. arXiv.

[12] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *ASPLOS*.

[13] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *ASPLOS*.

[14] Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, and Fred Chong. 2017. Memory Cocktail Therapy: A General Learning-Based Framework to Optimize Dynamic Tradeoffs in NVM. In *MICRO*.

[15] Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, and Michael F. P. O'Boyle. 2010. A Predictive Model for Dynamic Microarchitectural Adaptivity Control. In *MICRO*.

[16] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2014. Automated design of self-adaptive software with control-theoretical formal guarantees. In *ICSE*.

[17] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2015. Automated multi-objective control for self-adaptive software design. In *FSE*.

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

[19] Jason Flinn and M. Satyanarayanan. 2004. Managing battery lifetime with energy-aware adaptation. *ACM Trans. Comp. Syst.* 22, 2 (May 2004).

[20] Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. 2008. Quanto: Tracking Energy in Networked Embedded Systems. In *OSDI*.

[21] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. 2013. *Bayesian data analysis.* CRC press.

[22] Ashvin Goel, David Steere, Calton Pu, and Jonathan Walpole. 1998. SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit. In *2nd USENIX Windows NT Symposium*.

[23] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. [n. d.]. Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In *HPCA*.

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

[25] Henry Hoffmann. 2015. JouleGuard: energy guarantees for approximate applications. In *SOSP*.

[26] Henry Hoffmann, Anant Agarwal, and Srinivas Devadas. 2012. Selecting Spatiotemporal Patterns for Development of Parallel Applications. *IEEE Trans. Parallel Distrib. Syst.* 23, 10 (2012).

[27] Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. 2010. Application Heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *ICAC*.

[28] 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 *DAC*.

[29] T. Horvath, T. Abdelzaher, K. Skadron, and Xue Liu. 2007. Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control. *Computers, IEEE Transactions on* 56, 4 (2007).

[30] Connor Imes, David H. K. Kim, Martina Maggio, and Henry Hoffmann. 2015. POET: A Portable Approach to Minimizing Energy Under Soft Real-time Constraints. In *RTAS*.

[31] Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana. 2008. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *ISCA*.

[32] Syed Muhammad Zeeshan Iqbal, Yuchen Liang, and Hakan Grahn. 2010. ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems. *IEEE Comput. Archit. Lett.* 9, 2 (July 2010).

[33] C. Karamanolis, M. Karlsson, and X. Zhu. 2005. Designing controllable computer systems. In *HotOS*. Berkeley, CA, USA.

[34] David H. K. Kim, Connor Imes, and Henry Hoffmann. 2015. Racing and Pacing to Idle: Theoretical and Empirical Analysis of Energy Optimization Heuristics. In *CPSNA*.

[35] 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 (Jan. 2013).

[36] Etienne Le Sueur and Gernot Heiser. 2011. Slow Down or Sleep, That is the Question. In *Proceedings of the 2011 USENIX Annual Technical Conference*. Portland, OR, USA.

[37] B.C. Lee, J. Collins, Hong Wang, and D. Brooks. 2008. CPR: Composable performance regression for scalable multiprocessor models. In *MICRO*.

[38] Benjamin C. Lee and David Brooks. 2008. Efficiency Trends and Limits from Comprehensive Microarchitectural Adaptivity. In *ASPLOS*.

[39] Benjamin C. Lee and David M. Brooks. 2006. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *ASPLOS*.

[40] Matthew Lentz, James Litton, and Bobby Bhattacharjee. 2015. Drowsy Power Management. In *SOSP*.

[41] W.S. Levine. 2005. *The control handbook.* CRC Press.

[42] 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).

[43] J. Li and J.F. Martinez. 2006. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *HPCA*.

[44] Lennart Ljung. 1999. *System Identification: Theory for the User.* Prentice Hall PTR, Upper Saddle River, NJ, USA.

[45] C. Lu, Y. Lu, T.F. Abdelzaher, J.A. Stankovic, and S.H. Son. 2006. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. *IEEE TPDS* 17, 9 (September 2006), 1014–1027.

[46] 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

[47] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. 2013. Power Optimization in Embedded Systems via Feedback Control of Resource Allocation. *IEEE Transactions on Control Systems Technology* 21, 1 (Jan 2013).

[48] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Automated Control of Multiple Software Goals Using Multiple Actuators. In *ESEC/FSE*.

[49] John D. McCalpin. 1995. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE TCCA Newsletter* (Dec. 1995), 19–25.

[50] Nikita Mishra. 2017. *Statistical Methods for Improving Dynamic Scheduling and Resource Usage in Computing Systems.* Ph.D. Dissertation. https://search.proquest.com/docview/1928485902?accountid=14657

[51] Nikita Mishra, Connor Imes, Huazhe Zhang, John D Lafferty, and Henry Hoffmann. 2016. *Big Data for LITTLE Cores: Combining Learning and Control for Mobile Energy Efficiency.* Technical Report TR-2016-10. University of Chicago, Dept. of Comp. Sci.

[52] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. 2015. A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints. In *ASPLOS*.

[53] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. 2002. Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling. In *ICS*.

[54] Carl N Morris. 1983. Parametric empirical Bayes inference: theory and applications. *J. Amer. Statist. Assoc.* 78, 381 (1983), 47–55.

[55] Adam J. Oliner, Anand P. Iyer, Ion Stoica, Eemil Lagerspetz, and Sasu Tarkoma. 2013. Carat: Collaborative Energy Diagnosis for Mobile Devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*. ACM, New York, NY, USA, Article 10, 14 pages. https://doi.org/10.1145/2517351.2517354

[56] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Trans. on Knowl. and Data Eng.* 22, 10 (Oct. 2010), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[57] Paula Petrica, Adam M. Izraelevitz, David H. Albonesi, and Christine A. Shoemaker. 2013. Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems. In *ISCA*.

[58] Dmitry Ponomarev, Gurhan Kucuk, and Kanad Ghose. 2001. Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources. In *MICRO*.

[59] Raghavendra Pothukuchi, Amin Ansari, Petros Voulgaris, and Josep Torrellas. 2016. Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures. In *ISCA*.

[60] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. 2008. No "power" struggles: coordinated multi-level power management for the data center. In *ASPLOS*.

[61] R. Rajkumar, C. Lee, J. Lehoczky, and Dan Siewiorek. 1997. A resource allocation model for QoS management. In *RTSS*.

[62] Arjun Roy, Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. 2011. Energy Management in Mobile Devices with the Cinder Operating System. In *EuroSys*.

[63] Muhammad Husni Santriaji and Henry Hoffmann. 2016. GRAPE: Minimizing energy for GPU applications with performance requirements. In *MICRO*.

[64] 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 *SIGMETRICS*.

[65] Kai Shen, Arrvindh Shriraman, Sandhya Dwarkadas, Xiao Zhang, and Zhuan Chen. 2013. Power Containers: An OS Facility for Fine-grained Power and Energy Management on Multicore Servers. *SIGPLAN Not.* 48, 4 (March 2013), 65–76. https://doi.org/10.1145/2499368.2451124

[66] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. 2009. Koala: A Platform for OS-level Power Management. In *EuroSys*.

[67] Michal Sojka, Pavel Písa, Dario Faggioli, Tommaso Cucinotta, Fabio Checconi, Zdenek Hanzálek, and Giuseppe Lipari. 2011. Modular software architecture for flexible reservation mechanisms on heterogeneous resources. *Journal of Systems Architecture* 57, 4 (2011).

[68] Srinath Sridharan, Gagan Gupta, and Gurindar S. Sohi. 2013. Holistic Run-time Parallelism Management for Time and Energy Efficiency. In *ICS*.

[69] 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 (OSDI '99)*. USENIX Association, Berkeley, CA, USA, 145–158. http://dl.acm.org/citation.cfm?id=296806.296820

[70] Q. Sun, G. Dai, and W. Pan. 2008. LPV Model and Its Application in Web Server Performance Control. In *ICCSSE*.

[71] G. Tesauro. 2007. Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies. *IEEE Internet Computing* 11 (2007). Issue 1.

[72] Michel Tokic. 2010. Adaptive ϵ-Greedy Exploration in Reinforcement Learning Based on Value Differences. In *KI*.

[73] Stephen Tu and Benjamin Recht. 2017. *Least-Squares Temporal Difference Learning for the Linear Quadratic Regulator.* Technical Report 1712.08642v1. arXiv.

[74] Vibhore Vardhan, Wanghong Yuan, Albert F. Harris III, Sarita V. Adve, Robin Kravets, Klara Nahrstedt, Daniel Grobe Sachs, and Douglas L. Jones. 2009. GRACE-2: integrating fine-grained application adaptation with global adaptation for saving energy. *IJES* 4, 2 (2009).

[75] Greg Welch and Gary Bishop. [n. d.]. *An Introduction to the Kalman Filter.* Technical Report TR 95-041. UNC Chapel Hill, Department of Computer Science.

[76] Jonathan A. Winter, David H. Albonesi, and Christine A. Shoemaker. 2010. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *PACT*.

[77] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W. Clark. 2004. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *ASPLOS*.

[78] Weidan Wu and Benjamin C Lee. 2012. Inferred models for dynamic and sparse hardware-software spaces. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on.* IEEE, 413–424.

[79] Joshua J. Yi, David J. Lilja, and Douglas M. Hawkins. 2003. A Statistically Rigorous Approach for Improving Simulation Methodology. In *HPCA*.

[80] Wanghong Yuan and Klara Nahrstedt. 2003. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *SOSP*.

[81] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *ASPLOS*.

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

[83] Xiao Zhang, Rongrong Zhong, Sandhya Dwarkadas, and Kai Shen. 2012. A Flexible Framework for Throttling-Enabled Multicore Management (TEMM). In *ICPP*.

[84] Yanqi Zhou, Henry Hoffmann, and David Wentzlaff. 2016. CASH: Supporting IaaS Customers with a Subcore Configurable Architecture. In *ISCA*.

[85] Yuhao Zhu and Vijay Janapa Reddi. 2013. High-performance and energy-efficient mobile web browsing on big/little systems. In *HPCA*.