VaRTOS: Optimizing Application Utility with Variable Hardware

Paul Martin†, Lucas Wanner*, Puneet Gupta†, & Mani Srivastava†
†Department of Electrical Engineering, *Department of Computer Science
University of California, Los Angeles
Email: { pdmartin, wanner, mbs } @ucla.edu { puneet } @ee.ucla.edu

***Abstract*—Increasing hardware variability in newer fabrication technologies has spurred research in kind, spanning the entire spectrum of computing. One domain that stands to benefit greatly from research into combatting hardware variation is that of low power embedded systems and low power sensors. Systems tasked with performing sensing, routing, and inference tasks over extended lifetimes with limited energy reserves often resort to very liberal duty cycling, reducing the total active time spent on any given task. The lifetimes of such systems are largely dominated by leakage power in idle states—an entity that is greatly affected by hardware variation. We present VaRTOS, an architecture and corresponding kernel extensions that provide explicit treatment of both sleep and active power variations for duty cycled tasks running in real time operating systems. VaRTOS seeks to maximize application utility while converging on an optimal duty cycle both on a per-task basis and on a system-wide basis. In doing so, we provide results regarding online learning of instance-specific sleep power, active power, and temperature profiles on simulated hardware with demonstrated effects for several prototypical applications.**

I. Introduction

The emergence of low power wireless systems in past decades was followed closely by attempts at optimizing energy efficiency and power consumption to facilitate long lifetime sensing deployments. Research in this area was marked by many successes in the form of efficient time synchronization algorithms cite?, routing protocols cite CTP?, new sensing paradigms cite CS?, and the emergence of several widely-adopted operating systems for embedded systems (e.g., [1], [2]). Newer IC fabrication technologies have introduced several additional variables into the energy management game; in addition to power variation across temperature, power variation on a per-instance level has become a non-trivial factor [3], [4]. Figure 1 offers more intuition into the matter, showing the projected variation in idle and overall power as projected by IRTS for years to come cite here as well as recent variability results in the literature.

Considerable research has focused on exposing energy and power measurements or estimates to the developer and even to the end user. For example, techniques using linear combinations of instruction and performance counters [5], [6], activity monitoring strategies [7], and various battery discharge-based models [8] have been explored as tools for providing system-wide, application-specific, and even per-process granularities

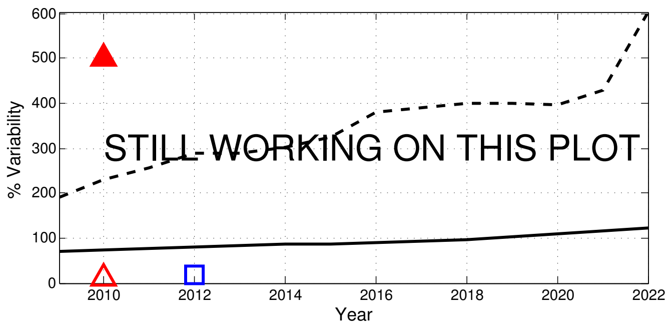

Fig. 1: IRTS projections for sleep and idle power variability with measurements from recent literature.

of energy usage. Improved techniques for accurately estimating and measuring power consumption will enable online power management in ways that were infeasible before, helping mitigate fabrication-induced per-instance chip variation as well cite here?.

Perhaps the most widely used and most effective strategies for extending the lifetime of energy-constrained systems are those based on controlling the ratio of active to idle time that the system is allowed, or *duty cycling* a system. Duty cycled systems take advantage of the disparity between active and idle power consumption, greatly increasing the lifetime of systems where latency and throughput constraints can be relaxed. Because of temperature and instance dependencies in power consumption, however, arriving at an optimal system-wide duty cycle ratio to achieve a lifetime goal given an energy constraint is difficult to do without empha priori knowledge of instance-specific power models and temperature statistics for the target deployment location [9]. Furthermore, applications involving more than one task introduce notions of fairness and utility—specifically, how should active processor time be distributed between each task so as to maximize the utility of the application and still meet the desired lifetime goal?

In this paper we explore the interplay between variable active and idle power consumption, deployment-specific temperature profiles, and multiple heterogeneous tasks with corresponding duty cycles. Specifically, we seek an answer to the question posed above; in an environment where power and temperature are measurable quantities, we seek an optimal strategy for distributing active processing time between arbitrary tasks so as to maximize application utility.

Our contributions include the following: (1) we demonstrate an efficient and simple method for online-learning of idle and active power curves which are, in general, nonlinear functions of temperature; (2) we distill the minimum statistics regarding temperature required for accurate system lifetime predictions; (3) we evaluate VaRTOS, a series of kernel extensions to the FreeRTOS operating system that provides explicit treatment and online optimization of multiple-task duty cycles with corresponding utilities; and (4) we evaluate the effects of VaRTOS on several prototypical applications, using a modified version of the QEMU simulation suite cite QEMU.

II. Related Work

Power management and longevity in wireless sensor networks and low power embedded systems in general has long been an active area of research. Traditional power management techniques have assumed that, while idle and active powers vary as a function of temperature, they remain uniform across instances. Of these variability-agnostic techniques, many have focused on the tradeoff between energy and utility or performance. For example, [10], [11] represent attempts at making quality energy-proportional and tunable. Specifically, [10] introduces an architecture that allows developers to specify multiple versions of functions whereby the operating system can sacrifice quality when possible to reduce computational costs. Similarly, [11] proposes tunable feature selection for wearable embedded systems, where less accurate feature computation can be used at the cost of inference quality. Similarly, [12] represents one of many efforts at using approximate computing to save energy where marginal losses in quality can be afforded.

More recent efforts have begun to consider hardware variability and methods for mitigating variation in software. These can roughly be divided into two camps: (1) low-level treatment of variability in hardware and in voltage and frequency scaling (e.g. [13]) and (2) application-level adaptations to meet power requirements. While low-level treatment of hardware variation is undoubtedly a necessary step forward for commercial processors, application- and process-level adaptations have proven effective methods for combatting variation in low power embedded systems and wireless sensor networks. Both [14] and [15] provide lifetime analyses for wireless sensor networks when considering variability power models, offering insights into what such systems stand to gain from explicit treatment of hardware variation. Garg et al. estimated that a 37% system lifetime improvement could be achieved through redundancy efforts that totaled a 20% increased deployment cost. To avoid cost increases, application quality or performance can be reduced as in [16], where video compression codecs are selected based on particular hardware variability ‘signatures’.

This work is closely related to that of Wanner et al. in [17]. There, the authors describe a method for calculating a system-wide optimal duty cycle ratio given known models for active and idle power as well as probability density functions for deployment temperatures. Here we provide an extension to the work in [17], showing methods for online learning of power models and providing notions of utility in multi-task applications.

III. Power Variability

As fabrication technologies improve and feature sizes decrease, hardware variation plays an increasingly important role in determine the power consumption and therefore lifetime of computer systems. While the large baseline in active power consumption relative to idle power consumption amortizes these variations to some degree ([17] cites a 10% variation in active power while [18] cites between 7% and 17% variation), the low baseline in idle power consumption renders it highly susceptible to fabrication-induced variations ([17] reports a 14x range in measured idle powers across 10 instances of ARM Cortex M3 processors).

Power consumption in energy-constrained embedded systems is often dominated by time spent in an idle state, waiting to sense, communicate, or perform a processing task. Consequently, the lifetimes of these systems can be wildly variant due to instance-to-instance variation in idle power. The result of inaccurate lifetime estimates obtained by improper power models can be one of two things: conservative designs can make sure that lifetime is met by adding a guard band to system duty cycle and thus underperforming, or removing these guard bands will result in higher performance while all subsystems (perhaps nodes in a network) are still running, but upon depletion of energy reserves by those processors with higher power consumption the entire system will again suffer. A processor-specific duty cycle can be assigned so that all lifetimes are met, but without a strategy for distributing resources to individual tasks on a processor there is no guarantee that these variations are handled elegantly and in a way that maximizes application utility.

IV. VaRTOS

In this section we present VaRTOS—a variability-aware architecture and corresponding extensions to the FreeRTOS cite here kernel for optimizing application-wide utility. In doing so, we will introduce models for heterogenous tasks that comprise an application as well as corresponding utility functions.

A. Task Modeling

We start first by introducing an *application* A as a set of *tasks* $\{T_1, T_2, \ldots, T_n\}$, where each task represents a periodic subprocess of A. It is assumed that a task T_i will add an increase in utility δu_i when given additional active compute time, $\delta T_{a,i}$. Here, u_i is a utility function specific to task T_i and analogous to quality of service (QoS). Whereas some efforts take the stance that the u_i should be provided by the developer ([10] for example), we advocate offloading the burden to the OS itself. In doing so, we will assume that u_i is a monotonically non-decreasing function of active computation time T_a, i or, equivalently, the duty cycle ratio specific to T_i denoted d_i. Tasks will offer varying degrees of utility for different d_i, and so some degree of user input is required to guide the optimization routine in distributing active time in an efficient manner. To incorporate this user input, we introduce the concept of a task *knob*.

Task Knobs: In order to tune the active time used per task and thus the task-specific duty cycle ratio d_i, we introduce the notion of task knobs denoted k_i, with the intuition that increasing k_i will increase $T_{i,a}$, d_i, and u_i. In practical terms, a task knob is a variable that will govern either (1) the period of a task or (2) the frequency with which a task is activated. We argue that the vast majority of tasks found in embedded applications will fall in one of these two classes, and those that require both frequency and period modulation can often be divided into two legal subtasks glued together with inter-process communications. For example, those that fall in class 1 include variable length sensing tasks, listening for inbound communication, and variable length processing chains. Those that fall in class 2 include variable frequency transmission, variable frequency sensor sampling, time synchronization handshaking, control and actuation events, and more.

Task knobs are created by passing a variable pointer to the OS, allowing direct manipulation of knobs by the optimization routine. In addition, the developer specifies a minimum and maximum knob value, $k_{i,min}$ and $k_{i,max}$. The value $k_{i,min}$ specifies the minimum amount of work T_i must do in order for it to be useful. Below this value, a task offers no utility. The value $k_{i,max}$ specifies a value at which increasing k_i further will yield only marginal or no added utility. As an example, a radio transmission task may be useless if it does not meet a certain latency requirement, but usefulness may plateau at some frequency governed perhaps by the physics and time response of the event being sensed.

Interpolating Utility: given $k_{i,min}$ and $k_{i,max}$ we construct a utility function $u_i = f(d_i)$ as a logistic (Sigmoid) function of the form

$$u_i(d_i) = \frac{1}{1+e^{-c_i d_i}}, \quad c_i \geq 0$$

We require only the convex portion of the Sigmoid function, choosing $u_i(d_i)$ to be of the particular form

$$u_i(d_i) = \frac{2}{1+e^{-c_i d_i}} - 1, \quad c_i \geq 0 \tag{1}$$

Here, c_i governs the convergence rate of u_i from the minimum utility to the maximum utility and is calculated as a function of $k_{i,min}$ and $k_{i,max}$ such that 99% of the utility has been reached by k_{max}. Increasing the percentage of $u_{i,max}$ realized by $k_{i,max}$ has the effect of arbitrarily steepening the utility curve. The constant c_i can then be calculated from Equation 1 as

$$c_i = \frac{-log\left(\frac{2}{\varepsilon+1} - 1\right)}{d_{i,max} - d_{i,min}}, \quad \varepsilon = 0.99 \tag{2}$$

Finally, each utility curve can be arbitrarily scaled by a priority scalar p_i for tasks with intrinsically higher utility than others. Figure 2 shows three example utility curves corresponding to three tasks with various priorities and duty cycle ranges (resulting from various $k_{i,min}$ and $k_{i,max}$).

Learning the $k_i \rightarrow T_{i,a}$ Relation: Because the developer has free reign to use the knob k_i for each task as desired, the function mapping k_i to active time $T_{i,a}$ and thus d_i is not known *a priori*. Instead, the transformation $\mathscr{K}_i$ that maps k_i to $T_{i,a}$ is

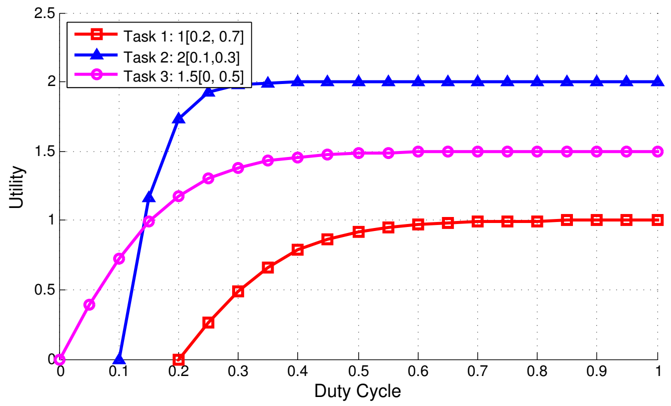

Fig. 2: Example utility curves. Task 1 has priority scalar 1 with useful range $d_i = [0.2, 0.7]$, task 2 with priority scalar 2 and so forth.

assumed linear and is learned through regression at runtime. Should the developer misuse k_i in a way that is nonlinear or results in non-increasing values of $T_{i,a}$, the optimization routine will suffer. Active time per task can be measured using hardware timer snapshots at the context swap level or other mechanisms described later in the text. Given a method for measuring $T_{a,i}$, $\mathscr{K}_i$ is arrived at by systematic perturbation of k_i within the range $[k_{i,min}, k_{i,max}]$. Specifically, k_i is repeatedly increased by $\Delta = \frac{k_{i,max} - k_{i,min}}{N}$ where N is the number of points in the regression, kept sufficiently low ($N = 3$ in our case) to minimize memory footprint. Between each perturbation in k_i, the task is allowed to run for a time period sufficiently long enough to capture active time measurements even for tasks with very infrequent activity. In the applications presented here, this supervisory period is set at 1 hour. Dividing active time accumulated per task by the supervisory interval yields task-specific duty cycle ratios, d_i.

B. System-wide Duty Cycle

Distributing active time per task assumes that we can afford a certain system-wide active time per supervisory interval, or equivalently that the system is governed by an optimal duty cycle d^*. Arriving at d^* is done in the same way as set forth in [17] with the exception that the active and idle/sleep powers (P_a and P_s) are learned at runtime. The optimal duty cycle d^* is calculated as a function of P_a, P_s, and $\boldsymbol{f_T}$, where $\boldsymbol{f_T}$ is the probability density function of the temperature for the location where the system is to be deployed. Given these quantities, d^* can be found as a solution to the optimization problem

$$d^* = \max \quad d \quad \text{subject to:}$$

$$\sum_T f_T \left[dP_a(T) + (1-d)P_s(T)\right] \leq \frac{E}{L} = \bar{P} \tag{3}$$

That is, maximize d such that the lifetime and energy constraints are met. Here, E and L are the energy and lifetime specifications and $\bar{P}$ is the average power needed to obtain lifetime L. The optimal solution to Equation 3 can be obtained algebraically as shown in [17]:

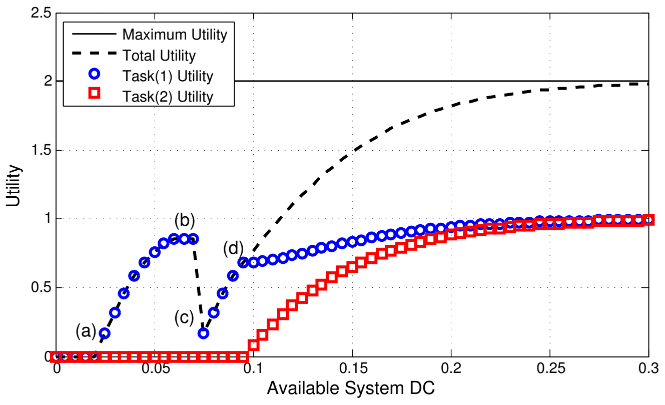

Fig. 3: optimizing shit

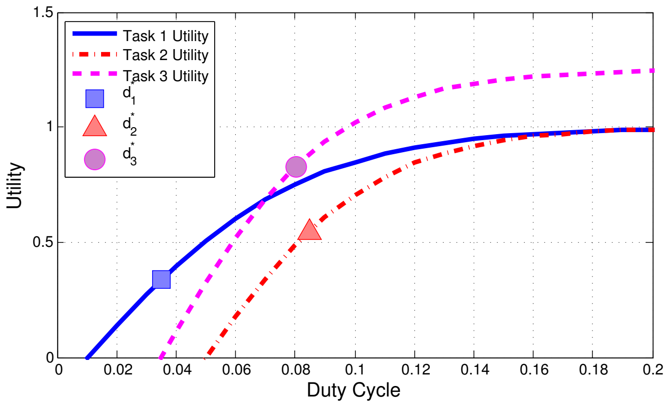

Fig. 4: optimizing shit

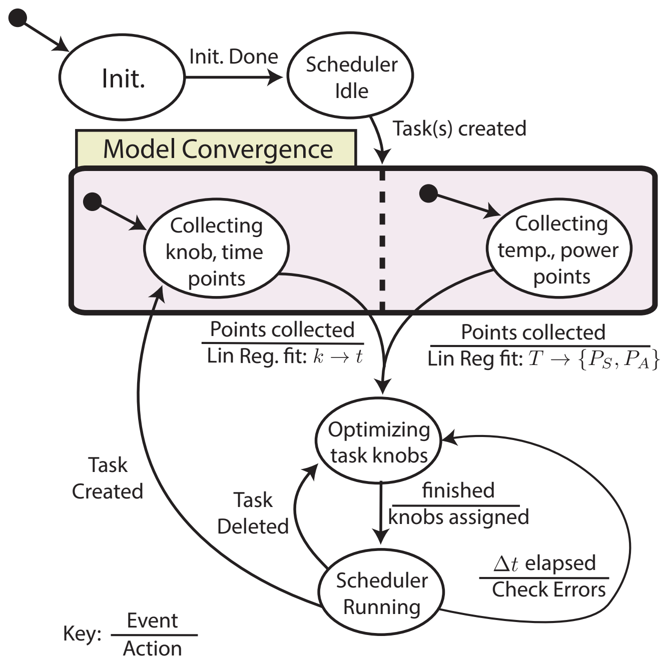

Fig. 6: state chart

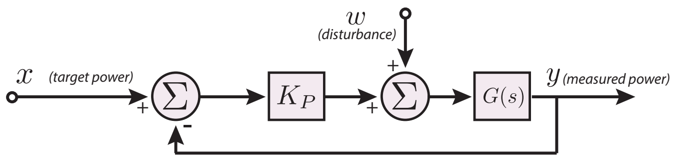

Fig. 7: state chart

$$d^* = \frac{E - L\bar{P}_s}{L(\bar{P}_a - \bar{P}_s)} \tag{4}$$

Where $\bar{P}_a$ and $\bar{P}_s$ are temperature-averaged power quantities.

C. Optimizing Utility

Because we have chosen u_i to be a logistic function, we can use a greedy approach when optimizing utility. The optimization routine will be a two step process: (1) attempt to assign the minimum duty cycle $d_{i,min} = \mathscr{K}_i(k_{i,min})$ needed for each task to yield utility in order of decreasing priority, and (2) continue distributing utility to the task

D. Instance Modeling

including sleep and active power, and musings on learning the temperature profile etc.

E. Operation

Talk about what happens in steady state, how we check for errors, what happens when errors become large, etc.

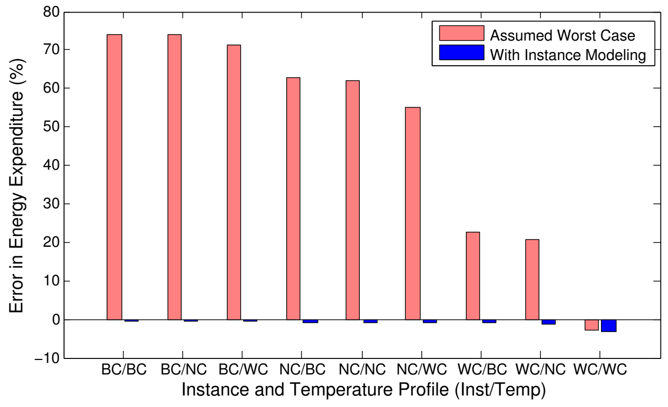

Fig. 10: App 1 energy comparison

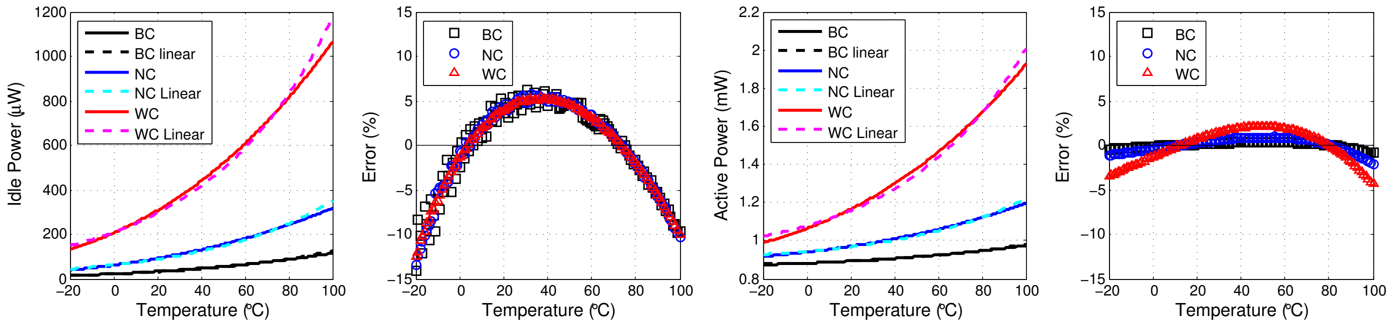

Fig. 5: Learning power....

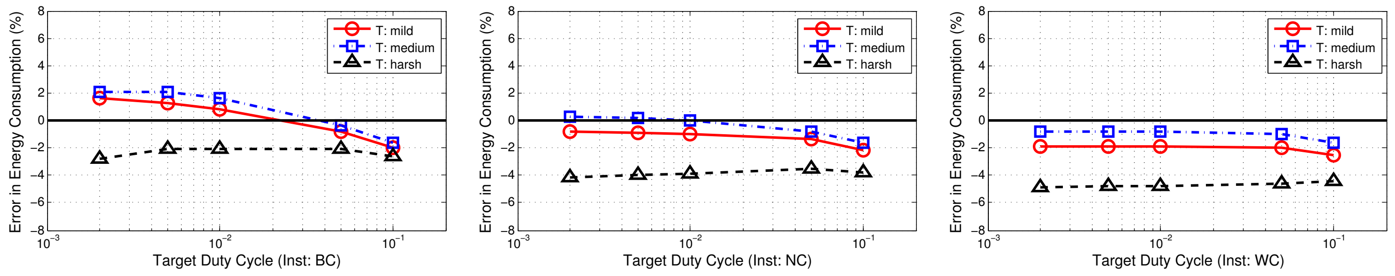

Fig. 8: App 1 results, no noise

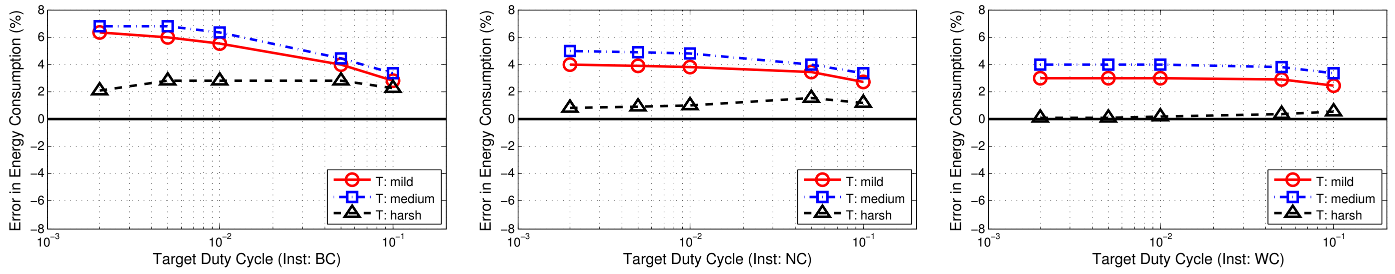

Fig. 9: App 1 results, no noise

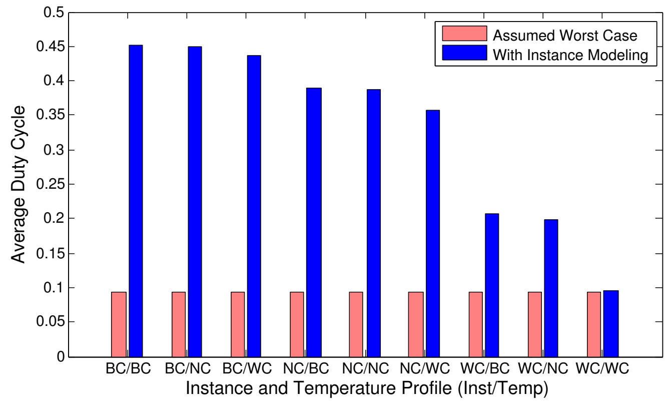

Fig. 11: App 1 dutycycle comparison

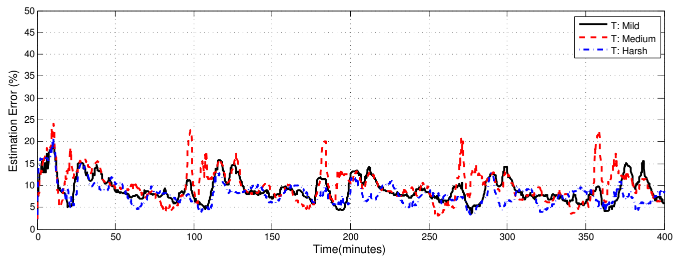

Fig. 12: App 1 dutycycle comparison

etc.

V. Evaluation

describe here the method of evaluation, introduce QEMU and VarEMU as tools for evaluation. Cite accuracy for varemu

A. *Multi-Sensor Applications*

Application 1: we have multiple sensors with various knobs fighting for resources with various utilities.

```
Data: $u_a(d) \forall a \in A,\ d_T^*$
Result: $d_a^*$
$d_{remaining} \leftarrow d_T^*$.
Assign min knob values:
for $a \in A$ do
    if $d_a(k_{min}) < d_{remaining}$ then
        $d_a = d_a(k_{min})$
        $d_{remaining} \leftarrow d_{remaining} - d_a$
    else
        stop
    end
end
Divvy up remaining duty cycle fairly:
while $d_{remaining} > 0$ do
    Find highest marginal utility:
    $maxM.U. \leftarrow 0$
    $numMax \leftarrow 1$
    for $a \in A$ do
        if $m(a) > maxM.U.$ then
            $maxM.U. \leftarrow m(a)\ numMax \leftarrow 1$
        else
            if $m(a) == maxM.U.$ then
                $numMax \leftarrow numMax + 1$
            end
        end
    end
    Increment task duty cycles:
    $d_{requested} \leftarrow \delta \cdot numMax$
    if $d_{requested} > d_{remaining}$ then
        $d_{requested} \leftarrow d_{remaining}$
    end
    for $a \in A$ do
        if $m(a) == maxM.U.$ then
            $d_a \leftarrow d_a + d_{requested}/numMax$
        end
    end
end
```

Algorithm 1: Greedy duty cycle optimization

B. Multi-Agent Applications

Application 2: we have multiple agents and we seek to fuse data and make some inference like localization. Basically sensors + radio

C. Digital Signal Processing Applications

basically sensors + (tunable DSP stages) + radio

VI. Conclusion

text

Acknowledgment

The authors would like to thank...

References

[1] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "Tinyos: An operating system for sensor networks," in *Ambient Intelligence (AmI '04)*. Springer Verlag, 2004.

[2] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, ser. LCN '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.

[3] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Design Automation Conference, 2003. Proceedings*, 2003, pp. 338–342.

[4] P. Gupta and A. Kahng, "Quantifying error in dynamic power estimation of cmos circuits," in *Quality Electronic Design, 2003. Proceedings. Fourth International Symposium on*, 2003, pp. 273–278.

[5] Y. Sun, L. Wanner, and M. Srivastava, "Low-cost estimation of sub-system power," in *Green Computing Conference (IGCC), 2012 International*, 2012, pp. 1–10.

[6] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 46–55, Jul. 2009.

[7] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: application energy metering framework for android smartphones using kernel activity monitoring," in *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, ser. USENIX ATC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 36–36.

[8] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 105–114.

[9] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava, "A case for opportunistic embedded sensing in presence of hardware power variability," in *Proceedings of the 2010 international conference on Power aware computing and systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8.

[10] W. Baek and T. M. Chilimbi, "Green: a framework for supporting energy-conscious programming using controlled approximation," in *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '10. New York, NY, USA: ACM, 2010, pp. 198–209.

[11] H. Ghasemzadeh, N. Amini, and M. Sarrafzadeh, "Energy-efficient signal processing in wearable embedded systems: an optimal feature selection approach," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED '12. New York, NY, USA: ACM, 2012, pp. 357–362.

[12] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J. Y. Chung, "Imprecise computations," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–94, 1994.

[13] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, 2008, pp. 363–374.

[14] T. Matsuda, T. Takeuchi, H. Yoshino, M. Ichien, S. Mikami, H. Kawaguchi, C. Ohta, and M. Yoshimoto, "A power-variation model for sensor node and the impact against life time of wireless sensor networks," in *Communications and Electronics, 2006. ICCE '06. First International Conference on*, 2006, pp. 106–111.

[15] S. Garg and D. Marculescu, "On the impact of manufacturing process variations on the lifetime of sensor networks," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, 2007, pp. 203–208.

[16] A. Pant, P. Gupta, and M. van der Schaar, "Software adaptation in quality sensitive applications to deal with hardware variability," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, ser. GLSVLSI '10. New York, NY, USA: ACM, 2010, pp. 85–90.

[17] L. Wanner, R. Balani, S. Zahedi, C. Apte, P. Gupta, and M. Srivastava, "Variability-aware duty cycle scheduling in long running embedded sensing systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, 2011, pp. 1–6.

[18] B. Balaji, J. McCullough, R. K. Gupta, and Y. Agarwal, "Accurate characterization of the variability in power consumption in modern mobile processors," in *Proceedings of the 2012 USENIX conference on*

Power-Aware Computing and Systems, ser. HotPower'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 8–8.