The Potential of Energy/Utility-Accrual Scheduling

Marcus Hähnel, Marcus Völp, Björn Döbel, Hermann Härtig

Operating Systems Group
Technische Universität Dresden
Dresden, Germany
Email: {*mhaehnel,voelp,doebel,haertig*}*@tudos.org*

***Abstract*—The long term vision of energy/utility accrual scheduling is to use all system resources in a way that is most beneficial to the system's users. For that, a mapping of user requests all the way down to system resources is required and, vice versa, the energy requirements of resources must be attributed to the corresponding user requests. However, despite the attractiveness of this general approach, the complexities involved in these translations are scary. Sketching our approach to energy/utility accrual scheduling, we argue in this paper that many complexities of traditional power models can be avoided if we consider the potential of a resource to generate utility rather than the utility generating operation. Introducing modes for the resources CPU and network, we found that the energy required to keep these resources operational is a good approximation of their overall energy demand.**

***Keywords*-resources, energy-efficiency, scheduling, modes**

I. Introduction

"Is it more purposeful to power up the 3G network to obtain additional bandwidth for a higher resolution video or should the remaing battery power better be spent to adjust the display's contrast and brightness setting to the environmental lighting?" Utility accrual scheduling gives a comprehensive answer to this question: do what is most beneficial for the users.

In economics, utility is a measure to express society satisfaction when given a certain combination of commodities. Utility accrual scheduling translates this concept into the realm of resource scheduling. Optimization criteria of classical resource scheduling, such as minimizing the number of deadline misses, maximizing throughput, or minimizing request latencies, are translated into maximizing the accrued utility of all activities.

Several utility accrual scheduling algorithms can be found in the literature (see Jensen et al. [1] for an excellent overview). Application areas range from datacenter cooling [2], grid [3] to wireless networks [4]. However, when it comes to energy as the primary resource, the complexities of translating consumed device energy back to users' requests negate most of the benefits obtained from utility accrual scheduling. For example, in [5] Pathak et al. report difficulties in attributing deferred network activities, tail power states and wakelocks of 3G networks to individual transmissions. In [6], Wu et al. introduce energy utility ratios as a linear relation between execution at a given voltage and frequency setting and consumed energy. However, clock-gated hardware resources, deep pipelines and cache hierarchies break this linearity even for relatively simple scenarios.

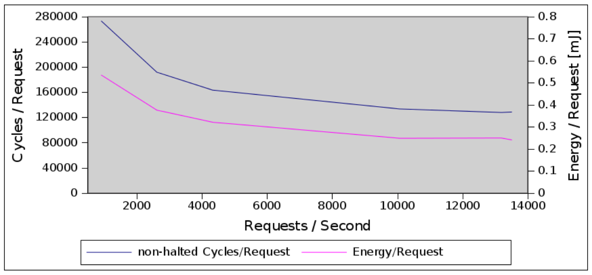


Figure 1: Request handling cost for a web server benchmark under varying load

Figure 1 shows the request handling costs in CPU cycles for ApacheBench [7] when stressing a single instance of the `lighttpd` web server. Mapping accrued execution time to utility, as advocated in [6], we would expect a constant service time and energy per request. The total consumed energy would then increase linearly with the number of requests offset of course by the server keepup power. However, Fig. 1 clearly shows that requests are serviced more efficiently under higher loads when caching effects and clock gating during idle times become effective. We shall return to this point in Section IV.

The key insight, which we convey in this paper, is our finding that the overall energy demand of system resources is often dominated by the energy required to keep these resources operational. By scheduling operation modes, that is, the potential of resources to generate utility, we avoid many of the complexities of traditional power models and put energy/utility accrual scheduling back on the horizon.

In the next section, we sketch our preliminary approach on energy/utility accrual scheduling and, in particular, on how perceived utility translates into resource demands. In Section III, we introduce *mode-utilities* as a manageable approximation of system energy demands. We evaluate the idea of mode-utility based energy modelling for the resources CPU and network in Section IV. We conclude this paper with directions for future work in Section V.

II. Towards Energy/Utility Accrual Scheduling

Viewed from some distance, energy/utility accrual scheduling works as follows: Every user of the system produces a statement of how happy she is when her desired mix of applications performs in a certain way. These statements can be made explicitly by turning appropriate knobs in the applications, or they can be gathered automatically from previously recorded profiles or by interpreting the current user behavior. The result is a favoring of application performances.

For example, let us assume a user switches focus to a video player, pushing other windows into the back. Interpreting this behavior as a sign that the user is more happy if she receives a better video quality, we adjust the weighting factor for this application and of all other applications of this user to obtain a new utility factor for the video player in the application mix.

The second ingredient to energy/utility accrual scheduling is a characterization of every application output according to how valuable this output might be for users. For our video player example, this is the observed video quality measured for example as the structural similarity index (SSIM) [8]. Mapping this quality metric into the interval $[0, 1]$ and weighting it as described above, we obtain a utility value for this application.

For applications to produce outputs of a certain quality, they have to execute in a manner that is in principle able to achieve the desired quality or a better one. We call these manners *application modes* and derive a compatibility criterion in Section III. In general, several users may simultaneously request outputs from a single application. In these situations, the application mode must be chosen such, that the application is able to produce the desired outputs at least in their desired quality. However, it is out of the scope of this paper to quantify whether application modes have the same scheduling simplifying effects as hardware *resource modues*.

To produce output of a certain quality, applications need resources either in the form of hardware resources or in the form of other applications, servers or operating-system functionalities, which we summarize as software resources. At the lowest level, only hardware resources remain. While the translation of resource requirements between different levels of the resource hierarchy is an interesting problem in its own right, in this paper we want to concentrate on the hardware level, assuming simple scenarios in the software layer.

Decoupling this hardware layer from the software stack allows us to create a scheme that has no strong dependency towards a specific hardware platform but where the translation between hardware and software layers serves as an abstraction between the resource requirements of the software and the energy characteristics of the hardware devices. Figure 2 illustrates this decomposition. Demands of applications and services with which users directly interact are ranked according to the utility they provide. Resource to resource-demand functions translate these requests down to demands of hardware resources, which consume their fraction of the system's overall energy demand. The `w` in the graph denotes a weight of a single user on the system, while the `(u,w)` tuples denote a tuple weight and utility which indicate a users preferences for a service.

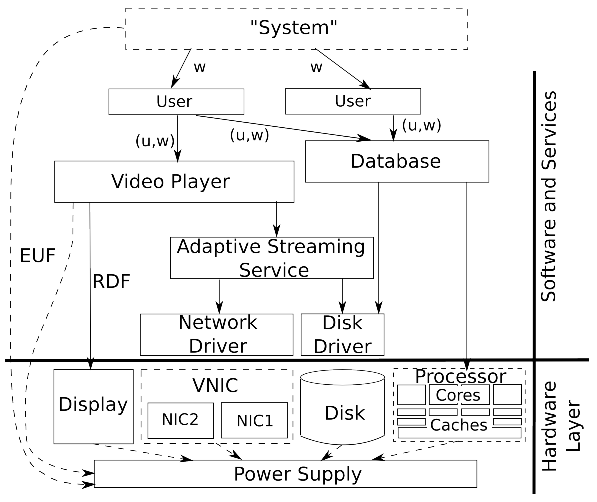


Figure 2: Decomposition of utility accrual scheduling into utility/resource, resource/resource-demand (RDF) and resource/energy functions

In general, the type of resources and the kind of requests can be specific for each client-server pair. However, we identified three metrics that are meaningful for all resources: latency, bandwidth and an admittedly ominous quality value, which is one (correct response) or zero (no / wrong response) for most resource requests. Scenarios where values except zero or one are purposeful include video players, where the SSIM is translated into the interval $[0, 1]$, or information processing, where accuracies significantly below 100% are sufficient.

As this paper focuses on low-level resources, detemining the precise resource to resource-demand functions is out of the scope of this paper. In fact, further work is required on characterizing more dynamic workloads such as database systems with changing data sets. For our evaluation, we collected measurement data to characterize how higher-level system components stress the underlying hardware resources.

For predicting future workloads, we plan to incorporate existing approaches [9] or, if this turns out to be too demanding, use heuristics such as the current application behavior.

Energy modelling is also discussed in a large body of related work. Most are concerned with building models that allow deducing the current energy consumption from a set of runtime measured properties (c.f. [10]). The recently introduced energy sensors (e.g., Intel's Running Average Power Limit (RAPL)) offers a more effective alternative to obtain energy demands [11] by observing and calibrating the power consumed by the monitored hardware resources on the very system we want to perform scheduling upon [12].

In the next section, we introduce our approach to mode based energy/utility functions (EUFs) for hardware resources.

III. Modes — A Heuristic for Utility

Our idea for modeling resource-level EUFs is based on the following two observations:

1) Most hardware resources support a number of different operation *modes* in which they generate utility if a given workload is placed on these resources; and
2) Keeping a resource in a certain mode consumes the bulk of the resource's total energy.

For example, CPUs can be run at different voltage and frequency levels, put in various sleep states or halted (e.g., with the help of special instructions indicating that no further work is to be expected). Displays can provide different degrees of brightness and contrast, and wireless network interfaces can emit signals at different strengths.

By our definition, any possible configuration of a hardware resource is considered a distinct mode. Running in such a mode enables the resource to provide a potential utility to its users. Following Observation 2, which we validate in Section IV, it is legitimate to approximate the energy consumed by a resource in a certain mode by a constant. Workload dependent fluctuations are insignificant with respect to this constant. We can therefore establish a mapping between the energy consumed to keep up the resource in a certain operation mode and the utility potential that this resource may generate in this mode. Hence, modes can be used as a simple heuristic for resource-level EUFs.

To calibrate our model, we cycle all resources through their available modes during startup and measure the energy consumption for these modes with the help of system built-in energy sensors. This already works for CPU energy consumption as described in [12]. We hope that in the future device vendors will make such information available at runtime as well. For now we rely on external energy models for those devices where no sensors are available.

Modes only represent the potential for applications to generate real utility. The task of the EUF-aware scheduler is therefore to map resource demands to the respective operation mode. The freedom it has in this choice gives room for further optimizations. For example, assume an application requests 2 billion CPU cycles. Depending on other constraints, such as real-time deadlines, the scheduler may achieve this by assigning the application a CPU at 2.0 GHz for one second or a 1.0 GHz clocked CPU for two seconds. It may even assign a 3.0 GHz clocked CPU for 0.67 seconds and halts this CPU for the remaining time. We are confident that this degree of freedom will allow us to combine our scheduling approach with others, such as Jensen's TUF-based scheduling [13].

As the above example shows, modes may be equivalent or at least subsume each other. However, there are also devices that exhibit incompatible modes. For instance, Pixel Qi's multimode display [14] can be operated either in transmissive or in reflexive mode, but obviously not in both simultaneously as this would require a "Schrödinger backlight", which can simultaneously be both turned on and off.

For finding compatible modes, we therefore arrange modes in a lattice[1]. where the bottom element $\perp$ stands for off and the top element $\top$ indicates an invalid mode. The scheduler may therefore follow this lattice to pick the least upper bound of all requested modes. In a lattice, least upper bounds are defined for all finite subsets of modes. The configuration is valid, if no resource is in mode $\top$.

Pathak et al. [5] pointed out that energy consumption cannot always easily be attributed to a running application and exemplify this with the help of a GSM modem's data transfers. The key observation we draw from their benchmark is that to avoid costly mode switches the modem remains in a high power mode even when no requests are sent. Other tail power effects arise from deferred garbage collection activities (e.g., in solid state disks). Lending ideas from real-time garbage collection [15], our approach is to overapproximate these costs and attribute a portion of these costs to each application request.

Finally, the system we envision our scheduler to run in, should also support adaptive applications that can provide varying levels of utility depending on constraints such as whether the system is currently running in battery mode or plugged into a power supply. Such adaptivity requires feedback between the scheduler and applications and we believe it fits into our model as well: Applications can specify multiple sets of resource requirements for different levels of provided utility. Furthermore, the scheduler may provide feedback to applications on the current system utilization and other external factors. This information may then be used by applications to select the adaptivity level they want to use.

IV. Evaluation

One of the essential building blocks of our energy/utility aware scheduler is our novel concept of utility-generating device modes. In this section we evaluate this approach for CPU and the network interface card, two of the most difficult resources to model when it comes to energy.

[1]A partial order $\leq$ over a set X forms a lattice if for every subset $S \subseteq X$ there is a unique least upper bound $\sqcup(S)$ in X.

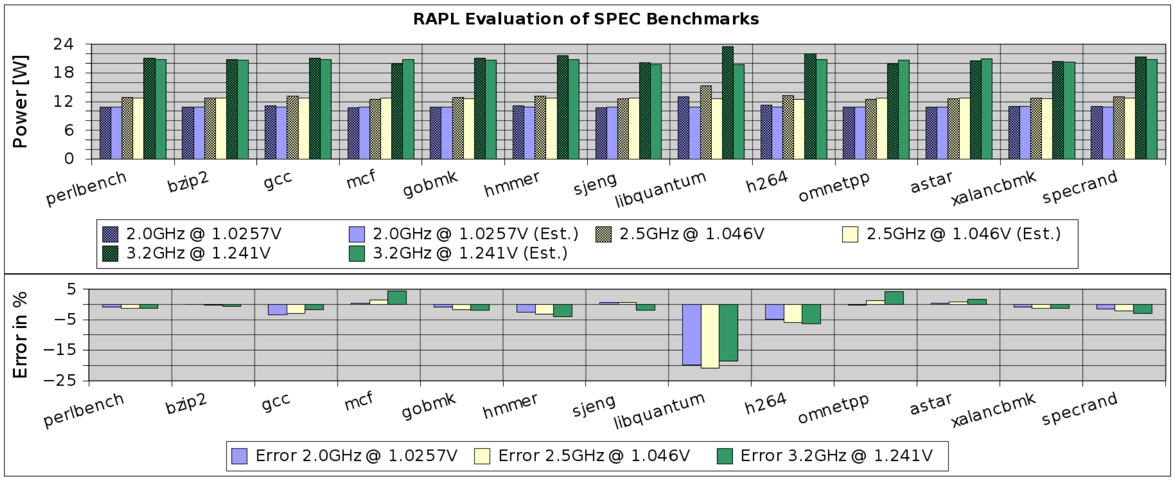


Figure 3: SPEC CPU 2006 Benchmarks at different frequencies

To support our claims that it is legitimate to approximate a resource's total energy consumption by a constant mode energy, we measured CPU power consumption of 3 different benchmark suites on an Intel i5-2400S with 2.5 GHz nominal frequency using the RAPL_PKG energy counter and a Linux 3.3.4 busybox installation, which we ran in an initial ramdisk. Linux was booted with `idle=halt` to prevent the kernel from entering deep C states during idle times. While our general concept also supports C-states, they require switching statistics and a more complex analysis to attribute how long the CPU did enter which of these states during our benchmarks. These statistics are available from recent Linux kernels, but do only increase complexity of the model, by introducing numerous additional modes. For a general evaluation we decided to keep the number of modes low to make the concept more easily tangible. In all benchmarks only one CPU core was active. All others had been disabled in the BIOS. An extension of our results to modes where more than one core is active is straight forward. In addition to the nominal frequency, we used ACPI frequency scaling to also measure the 2.0 GHz level and the 3.2 GHz turbo mode. The turbo mode results have to be taken with a grain of salt, because turbo overclocks the CPU only for short burst intervals and clocks down again when this is no longer safe. We did not evaluate any statistics on how long turbo remains at which frequency.

Figure 3 shows the results for the SPEC CPU 2006 integer benchmarks (CINT 2006). Shown is the measured power (shaded bars) and our approximation based on the fraction of non-halted vs. halted clock cycles (plain bars) for the three different frequencies (2.0 GHz, 2.5 GHz and 3.2 GHz (turbo)). For our approximation, halted clock cycles were accounted with the idle power of the CPU 7.5 W, while non-halted cycles were accounted with the help of a reference benchmark. We crafted this reference benchmark to put as much pressure on the interger units of the CPU as possible by executing a tight loop of independent integer instructions that completely fits into the caches. We did not stress the floating point units, SSE and AVX or the cache subsystem as we regard these as different resources. While this method will not yield the highest possible power consumption of the core, we found it to be a good approximation for Spec CINT. The bottom part of Fig. 3 substantiates this point by showing the relative error between our approximation and the measurements. With the exception of libquantum, the error is within a $\pm 5\,\%$ margin, with most benchmarks being as close to $\pm 2\,\%$. The cache analysis performed by Jaleel [16] indicates that libquantum has a workingset of 32MB and benefits only from a cache of this size or larger. This would indicate that a large number of cache misses are responsible for the unusual energy behavior of this application.

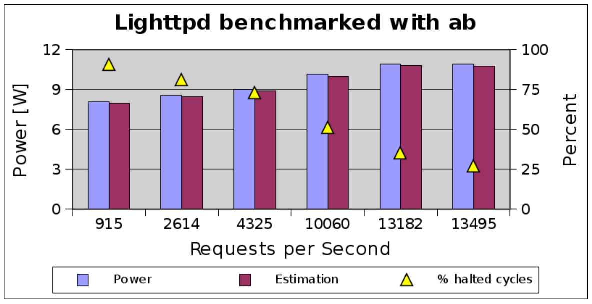


Figure 4: `Lighttpd` with different amounts of requests per second

These cache misses would constitute increased stress on the memory and cache subsystems and could thus be again regarded as a seperate mode. For our initial evaluation we decided to not incorporate a memory mode, but leave a more complete model of different CPU parts for future work

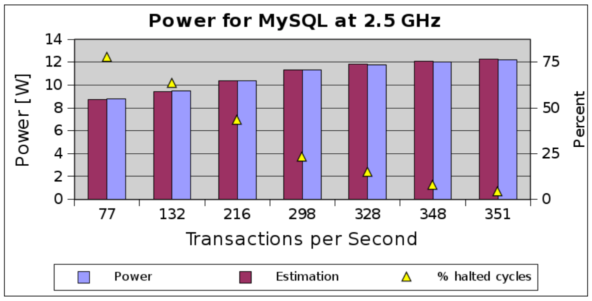


Figure 5: MySQL with varying transactions per second

Figures 4 and 5 show the results of our approach in two application scenarios: servicing 284 byte static webpages from Apache Bench (ab) with the help of the `lighttpd` web server (Fig. 4) and the MySQL-based OLTP benchmark from the sysbench suite with max-requests set to 50000 (Fig. 5). Shown is the consumed power for different request rates and the fraction of halted cycles. The prediction error is again within $\pm 2\,\%$.

In addition to these cycle based evaluations, we also performed an estimation based on the number of transactions per second (TPS). The previous benchmarks indicated a transaction handling time ($CPTS$) of 6.72 million clock cycles, an idle power P_{idle} of 7.639 W and a tight loop power P_{tight} of 12.5654 W at 2.5 GHz. Using 3 threads, we generated 176.75 TPS from which we calculated the ratio of used clock cycles as

$$r = \frac{TPS \cdot CPT}{f}$$

so that the equation

$$P_{est} = P_{idle} * (1 - r) + P_{tight} * r$$

gives us the estimated CPU power of 9.98 W. This confirms our measured energy of 9.86 W with an error of only 1.2 %.

To show that modes are also applicable for predicting a typical server application mix, we ran `lighttpd` and MySQL at the same time and on the same server. We addressed the challenge that web request per second (RPS) are not linearly correlated with cycles per request (CPR) with a profile where this translation was recorded. This profile was recorded over multiple sample runs and recording the number of running CPU cycles consumed and the request rates of the clients. From these we created a model that translates requests per second to CPU cycles. We recognize that this model is rather crude, but, as stated above, the prediction of resource usage is out of the scope of the paper and already covered by others [9] and further research is already ongoing in that area.

The estimation of the total power consumption is

$$P_{est} = P_{idle} * (1 - r_{sql} - r_{web}) + P_{load} * (r_{web} + r_{sql})$$

where

$$r_web = \frac{RPS \cdot CPR}{f}$$

With 10.6 W, our estimation for servicing 3000 web requests and 150 SQL transactions per second is only 1.2 % off the measured power of 10.47 W.

The above results confirm our approximation with an error margin of $\pm 5\,\%$ for most analysed applications with the exception of libquantum.

As a further device we also want to present our early results on the energy characteristics of an Intel Gigabit Ethernet card (EXPI9301CTBLK). To obtain these characteristics we instrumented a PCI-Express riser card.

We executed a UDP bandwidth benchmark (iperf) using this instrumented NIC, both in send and receive mode.

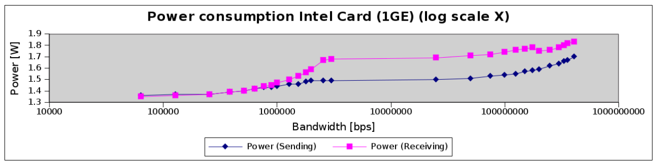
Figure 6 shows the correlation between bandwidth and energy consumption we measured. Figure 6a gives a complete detailed overview of the characteristics. Please mind the logarithmic x-Axis. It shows that the card basically combines two sets of energy characteristics: One for the low bandwidth range and a second in the middle to high bandwidth range. Figures 6b and 6c show the linear representation of the whole graph and the low bandwidth section respectively. We see, that when only looking at the respective characteristics, energy consumption scales nearly linearly.

As with the CPU, we attribute this linear increase with bandwidth to a combination of two modes: idle, and transfering, and attribute the efficiency gain in the mid to high level to a reduced number of wakelocks.

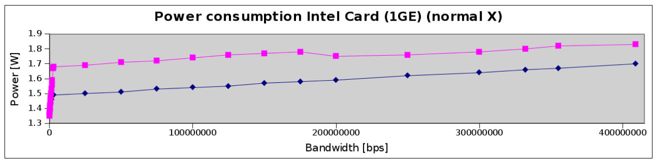
At this point we want to note, that the energy characteristics of network devices vary greatly between even different generations of the same card series, as well as the kernel driver version under investigation. Some of our network cards showed no variations of energy characteristics at all, displaying a static energy consumption that did not change beyond the margin of error after the link was established, irrespective of the actual usage of the link.

V. Conclusions & Future Work

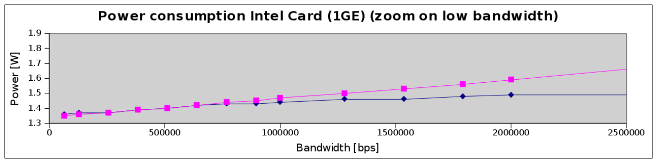
This paper introduces mode energy and the potential utility that a resource generates when operating in a certain mode as a manageable approximation for utility-accrual energy-efficient scheduling. We evaluated our approach with the help of the Spec CINT 2006 benchmarks and in two real world scenarios showing an error margin of $\pm 2\,\%$ for most applications and of $\pm 5\,\%$ for all except libquantum. We have also shown early network card results that promise compatibility of these devices with our modes model as well. The calibration of mode energies can be done by simple microbenchmarks using on-chip energy sensors such as RAPL. We confirmed our claim that mode energy dominates the overall energy consumption of a resource and that therefore utility accrual scheduling should focus on scheduling the



(a) Logarithmic scale



(b) Linear scale



(c) Low bandwidth in linear scale

Figure 6: Energy characteristics of the EXPI9301CTBLK Intel Gigabit Ethernet card in different scales

potential to generate utility rather than the complex utility generating workloads.

In future work we plan to extend our approach to further resources like disks and the display. The results of our preliminary analyses make us confident that our approach extends also to these resources.

Acknowledgment

We want to thank Dr. Waltenegus Dargie for providing the measurement instrumentation for the network card. This work was partially funded by the German Research Concil (DFG) through the Collaborative Research Center CRC 912 "Hightly-Adaptive Energy- Efficient Systems" (HAEC) and the Special Purpose Program "Depend- able Embedded Systems" (SPP 1500).

References

[1] B. Ravindran, E. D. Jensen, and P. Li, "On recent advances in time/utility function real-time scheduling and resource management," in *International Symposium on Object-Oriented Real-Time Distributed Computing*, May 2005.

[2] R. Das, H. Hamann, J. Kephart, and J. Lenchner, "Utility function-driven energy-efficient cooling in data center," in *International Conference on Autonomic Computing (ICAC)*, June 2010.

[3] C. Li and L. Li, "Utility-based scheduling for grid computing under constraints of energy budget and deadline," *Computer Standards and Interfaces*, vol. 31, no. 6, pp. 1131 – 1142, 2009.

[4] I.-H. Hou and P. R. Kumar, "Utility-optimal scheduling in time-varying wireless networks with delay constraints," in *Symposium on Mobile AdHoc Networking and Computing*. Chicago, Illinois, USA: ACM, Sept 2010.

[5] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM Europ. Conference on Computer Systems*. New York, NY, USA: ACM, 2012, pp. 29–42.

[6] H. Wu, B. Ravindran, E. D. Jensen, and P. Li, "Energy-efficient, utility accrual scheduling under resource constraints for mobile embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 3, pp. 513–542, Aug. 2006.

[7] "Apache HTTP server benchmarking tool," 2012. [Online]. Available: http://httpd.apache.org/docs/2.2/programs/ab.html

[8] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600 –612, april 2004.

[9] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload analysis and demand prediction of enterprise data center applications," in *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*, ser. IISWC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 171–180. [Online]. Available: http://dx.doi.org/10.1109/IISWC.2007.4362193

[10] D. C. Snowdon, S. M. Petters, and G. Heiser, "Accurate on-line prediction of processor and memory energy usage under voltage scaling," in *Proceedings of the 7th International Conference on Embedded Software*, Salzburg, Austria, Oct 2007, pp. 84–93.

[11] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *Proceedings of the 2011 USENIX Annual Technical Conference*, ser. USENIX ATC'11. Berkeley, CA, USA: USENIX Association, 2011.

[12] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using RAPL," in *SigMetrics/GreenMetrics*. London, UK: ACM, June 2012.

[13] U. Balli, H. Wu, B. Ravindran, J. S. Anderson, and E. D. Jensen, "Utility accrual real-time scheduling under variable cost functions," *IEEE Trans. Computers*, vol. 56, no. 3, pp. 385–401, 2007.

[14] PixelQi, "Pq 3qi-01 reflexive/transmissive color screen module," May 2012.

[15] R. Jones, A. Hosking, and E. Moss, *The Garbage Collection Handbook: The art of automatic memory management*. Wiley, August 2011, ch. 19, pp. 375 – 416.

[16] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation," July 2012. [Online]. Available: http://www.jaleels.org/ajaleel/publications/SPECanalysis.pdf