Selected Chapters of System Software Engineering

Energy-Aware System Software

Department of Computer Science 4 — Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

Proceedings

— July 2013 —

TABLE OF CONTENTS

Foreword

Timo Hönig <thoenig@cs.fau.de>
Christopher Eibel <ceibel@cs.fau.de>

Department of Computer Science 4
Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

Today, energy saving techniques are relevant for all computing systems: sensor nodes, mobile computing systems such as smart phones and tablet computers as well as Cloud Computing infrastructure. All these systems rely on energy-aware system software for various reasons. While the necessity is obvious for battery-operated systems, other systems are required to satisfy thermal limits, for example. Various approaches for performing work energy-efficiently exist: hardware energy saving features, energy-efficient operating system components, energy profiler and custom runtime environments.

By means of current research papers, this seminar presents different areas of energy-aware system software and discusses their approaches. Each of the five students who participated in the seminar chose one topic at the first seminar date. Beyond that, an introduction to the field of energy-aware computer systems and details on how to write a paper and how to give a presentation were given by the organizers. Furthermore, organizational matters were clarified. Each attendee had to write a paper in ACM two-column style with at least six pages and to give a 40-minute presentation. Each participant was either supervised by Timo Hönig or Christopher Eibel, who both reviewed the papers and presentation slides of their respective supervisees. At each seminar a different talk was scheduled. One week before each presentation, the corresponding draft version of the paper was sent to all participants for preparation so that all attendees were able to take part in further discussions at the seminar.

All seminar dates that were not reserved by students were filled with presentations held by the supervisors. One appointment was reserved for giving the students an introduction on how to read and understand research papers. Moreover, the general procedure of conferences and workshops was explained, for example, by giving insight into different types of reviewing processes. For one seminar, the attendees had to read the paper *"The Forgotten 'Uncore': On the Energy-Efficiency of Heterogeneous Cores"*, which was discussed at first. Afterwards, the digital recording of its original presentation at the USENIX ATC conference in 2012 was presented. The seminar was rounded up with details about the SEEP framework, including a live demonstration and details about the process of writing and submitting the paper to 4th Workshop on Power-Aware Computing and Systems (HotPower) in 2011.

Components for Energy-Efficient Operating Systems

Clemens Lang
Friedrich-Alexander University Erlangen-Nuremberg
clemens.lang@fau.de

ABSTRACT

The proliferation of smartphones and tablet computers has raised awareness of an important aspect of modern system design: energy usage can no longer be neglected when designing systems. Advances in battery technology are lagging behind other features like clock speed, memory, storage or bandwidth [3], which makes energy consumption a major issue in today's systems. Not only portable devices, but also data centers profit from energy-aware systems [1, 4].

Operating systems should not ignore energy consumption but use hardware-provided power saving features as efficiently as possible. This document outlines the components used in operating systems to perform efficient power management. It also attempts to give insight into the complexity of power management and highlights problems.

1. INTRODUCTION

Comparing the growth rates of battery capacity to those of clock frequency and memory yields alarming results: Table 1 shows that in the timespan it took to raise clock speeds to a hundredfold, battery life has only increased by a factor of ten. The comparison of RAM and battery life is even more steep.

Year	Clock Speed	RAM	Battery Life
1981	1	1	1
1991	4	512	2
2001	187.5	65535	4
2010	1200[1]	262144	10

Table 1: Technology trends of mobile devices: Features in multiples of their value in 1981, adapted from [3].

In contrast to memory management, which has been a standard component of operating systems for decades, energy consumption was largely unconsidered in software design before 1998 [9]. In general-purpose operating systems, energy efficiency can be increased by adjusting power consumption of a process at runtime. Predicting or measuring and extrapolating the power usage of a running program is a requirement for good power management decisions. For obvious reasons, prediction should add as little overhead on energy consumption as possible. Interpreting or analyzing code is thus a method out of question in online approaches to power management – we need different tools to achieve the necessary precondition. Furthermore, it is required to gain insight into where and how power is used in a computer; on a software level, that means finding out which behavioral patterns are the most energy inefficient ones. This does, however, not mean that other energy consumers in the system can be ignored if we want to minimize the power consumption of the whole system.

The following section will give a brief overview on where and how power is used in CPU and memory. Methods to estimate the power usage at runtime without specialized hardware will be highlighted. Section 3 lists mechanisms available in common hardware to reduce the power usage. The following section discusses operating systems' approaches to minimizing energy consumption by using policies and a number of challenges with software-based power management methods, giving insight into the complexity of power management. Section 5 covers implementation details, before the last section concludes this overview.

[1] in two cores

2. MEASURING POWER CONSUMPTION

Power in semiconductor chips is consumed when current is flowing due to either leakage (i.e., power dissipation) or due to loading or unloading of capacitors caused by a transistor switch [10]. Power dissipation depends on static parameters like voltage and time. The power usage also consists of a dynamic part that is caused by the switching of transistors. Assuming parts of the chip with high switching frequencies account for a significant part of energy consumption defines a region of interest when searching for energy-demanding areas in the system.

Although static power usage might seem irrelevant at first, it can still be beneficial to model it to rule on scheduling and power management decisions. Since static power consumption depends on the voltage, systems using dynamic voltage scaling (DVS) should not neglect it when optimizing energy usage.

To model dynamic power consumption accurately, a precise understanding of where power is used dynamically is crucial. Regions of interest, among others not being discussed in this article, are the *central processing unit* (CPU), caches, a *memory management unit* (MMU), and *dynamic random access memory* (DRAM).

The CPU's high switching frequency suggests a considerable contribution to power usage. The instructions being executed might influence the dynamic power consumption of the processor; this will be discussed in greater detail on the following pages.

Depending on the associativity and the frequency of references, caches also contribute to dynamic power consumption.

One would expect applications heavily using the cache to show higher energy consumption than others keeping their data in CPU registers. Due to the caches inside an MMU like the *translation lookaside buffer* (TLB), significant power usage can also be expected in memory management.

Random access memory is responsible for another considerable share of power consumption. On the one hand, this is caused by the complex mechanisms involved in accessing DRAM. On the other hand, the period refresh needed for dynamic RAM is reflected in an increased static power consumption. While one expects the static part to be independent of the instructions being run, memory-intensive applications could cause higher dynamic energy usage.

Note that this enumeration is not exhaustive. Peripheral devices, storage, I/O, cooling and network cards further contribute to power consumption and may each offer their own power saving mechanisms but are not discussed in this article.

To effectively optimize efficiency by reducing dynamic power consumption, a relation between certain behavioral patterns of software (like a series of memory accesses, or a CPU-bounded computation) and the power consumption at different power saving settings is required. This data can either be supplied by the manufacturer or be measured using a set of benchmarks. Note that current consumption measurement equipment is not integrated into off-the-shelf computers; the data can thus not be acquired online for the system at hand but must be gathered ahead of time. Due to power consumption being very hardware-specific, we have to assume the results will not generalize well enough to be useful for other setups.

Both Weißel et al. and Snowdon et al. measure energy consumption of the complete system they are trying to optimize. The former used a shunt resistor to measure the current flowing into their test system, an Intel IQ 80310 PCI board rated at 3.3 V. The static power consumption was measured while the test board was idle. A commercial AC power meter between the machine's power plug and the wall socket measured power for Snowdon et al. While both test setups did measure the power consumption at the point were it should be minimized, detailed statistics for parts of those systems (e.g., CPU, memory, I/O) have not been generated. The recent development to integrate chips with each other using the package on package approach or by merging the chips completely further increases the difficulty of separate measurements. On-chip mechanisms to gauge power consumption could solve this problem and simplify per-component measurements.

2.1 Event Counters

To tell different software behaviors apart assuming the software is unknown ahead of time, event counters can be used. Commonly used for performance analysis, event counters are hardware-provided counters for events, such as cache misses, cycles, or memory accesses. The number of countable events depends on the processor and is usually in the range of hundreds [5]. However, only a few events can be measured simultaneously due to the relatively low number of event counters. Typical numbers of available counter registers are single-digit to low two-digit numbers, e.g., between 2 and 6 for ARM11 and the ARM Cortex series of processors [6].

Since the number of events exceeds the number of available counters, a subset of events must be chosen. To determine the ideal set of events, a series of measurements can be used: running a number of benchmarks with known behavior at different power saving settings with different subsets of event counters selected generates data that can be used to compute the correlation between counted events and their corresponding power consumption. The events showing a high correlation are good candidates to estimate the energy usage of a process when dedicated energy measurement hardware is not available.

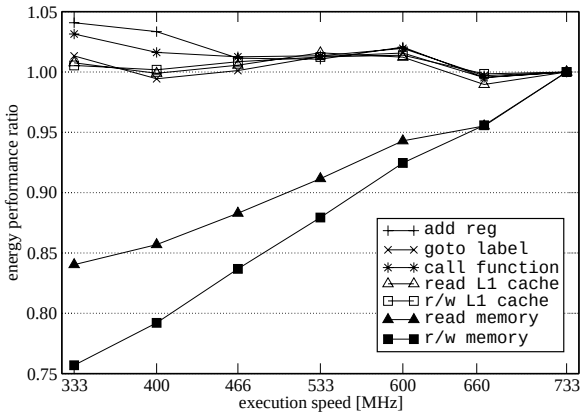

Figure 1: Energy consumption for several benchmarks relative to the energy consumption at full clock speed. From [13].

To target highest efficiency in power usage, the *energy performance ratio* must be minimized. An energy performance ratio of r % at a certain energy saving setting means the task needs r % of the energy it would have needed at full clock speed. Research shows CPU-bound tasks run at high efficiencies regardless of clock speed, while memory-intensive software is up to 25 % more efficient at lower execution speeds in a test setup used by Weißel et al. in 2002. Figure 1 shows this: the five benchmarks "add reg", "goto label", "call function", "read L1 cache" and "r/w L1 cache" are efficient at all tested clock speeds, while the memory-bound tasks "read memory" and "r/w memory" have a lower energy performance ratio (i.e., higher efficiency) at lower clock speeds. Weißel et al. attribute the savings to the slow response time of the memory and the cycles the CPU needs to stall and wait for the results of the memory access. Since the gap between memory speed and CPU clock speed has increased even further since 2002, this effect is more relevant than ever before. Lowering the CPU frequency relative to the memory frequency reduces the number of cycles wasted, thus increasing efficiency. Note that Weißel et al. only discuss dynamic voltage scaling in theory, but have not measured energy usage with DVS in effect due to missing hardware support for voltage scaling.

2.2 The Performance Energy Trade-off

It is not sufficient to minimize the energy performance ratio in order to build a power-efficient but fast system. We have seen in Figure 1 that CPU-bound tasks are almost equally efficient at any power saving setting while memory-bound tasks are more efficient at lower clock speeds. Considering only these results, the best strategy to save power would be always running at the lowest possible frequency. However, performance and energy usage are closely linked: Figure 2 shows the performance of a subset of those benchmarks in

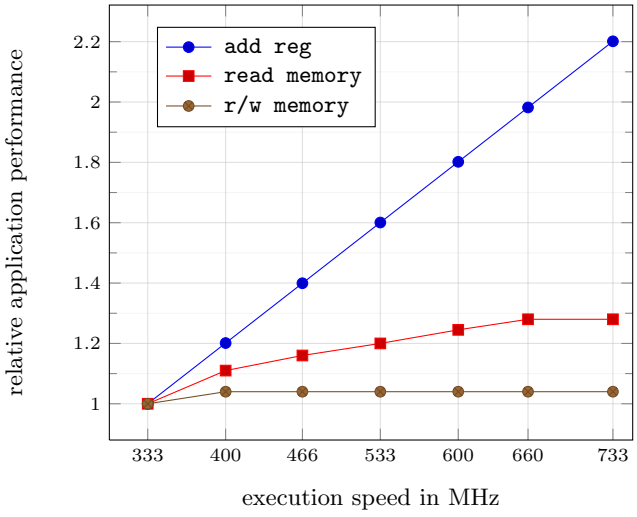

Figure 2: Normalized application performance at different clock speeds. Adapted from [13].

multiples of the execution time at the lowest clock frequency. We can see that the CPU-bound task "add reg" (and the other CPU-bound tasks omitted in this graph) would be slowed down significantly by running at lower clock speeds, leading to a worse user experience in interactive systems.

Since power management decisions also affect the user experience, it makes sense to allow the user to adjust the power management behavior. A simple approach to limit the effects of power saving on user experience is establishing an upper bound on performance loss under power saving. Weißel et al. chose a maximum loss of 10 % in [13], leading to energy savings of up to 15 % compared to execution at optimal performance.

3. POWER SAVING MECHANISMS

Assuming accurate prediction of the behavior of a task, how can power actually be saved? Which mechanisms are offered by today's hardware to reduce power consumption? This section will explain common techniques and discuss power management heuristics proposed in research.

3.1 Dynamic Frequency Scaling

Modern processors' clock frequencies can be adjusted at runtime. Since less switching occurs at lower frequencies, less power is consumed. For example, Intel processors that came to market after the Pentium M processor support *Enhanced Intel Speedstep®Technology* that will scale the core frequency by writing to a machine-specific register [5]. Dynamic voltage scaling is a de-facto standard to save energy [13, 8]. Frequency scaling is often combined with voltage scaling to achieve a higher impact on power consumption.

3.2 Dynamic Voltage Scaling

Dynamic voltage scaling (DVS) is a technique where the supply voltage of the CPU is changed during execution. Since the used energy is proportional to the square of the supply voltage, i.e.,

$$E \propto V^2, \tag{1}$$

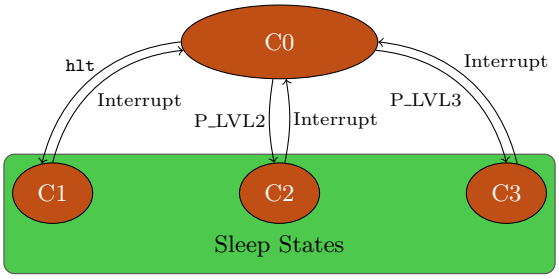

Figure 3: C-states as defined by ACPI. Adapted from [2].

lowering the voltage can reduce the energy consumption significantly [10].

Both dynamic frequency scaling and dynamic voltage scaling are often combined into dynamic frequency and voltage scaling (DVFS), because most processors only support lower voltages at lower clock speeds. Both methods involve a switching time that might be non-negligible and thus has to be taken into account when making power saving decisions. A particularly slow example is the AMD Opteron 246 processor, where voltage scaling takes up to 2 ms when operating within the specifications. This is well outside common switching times in the range of 10 μs to 140 μs Snowdon et al. found for other CPUs [11].

3.3 Sleep States

Modern systems provide several sleep states used for power management. The *Advanced Configuration and Power Interface* (ACPI) standard defines such states for different parts of the hardware. For the short-term power management discussed in this article, the power states of the processor are most relevant. The ACPI specification denotes these states C0, C1, C2, C3, . . . , C*n* [2]. While in state C0, the processor executes instructions. Using the `hlt` instruction, the processor can be put into the *C*1 power state, in which it ceases to execute instructions but maintains execution context and caches. All higher-numbered states are optional. In C-state 2, processors keep their context and caches (which also implies continuation of the cache-coherence protocol) but consume less energy than in C1. C2 is the first state where wakeup delays may be non-negligible. In C3 the processor consumes even less power by handing off cache-coherence to a different entity or flushing its caches completely. Figure 3 illustrates the available C-states and their transitions. Switching delays for these states are available from ACPI to be used by power management policies. Note that switching is only possible between C0 and any other C-state.

Implementations of power management components in operating systems should be aware that hardware may support more than the states defined by the ACPI standard, such as the sub-C-states in Intel CPUs [12, 5], possibly leading to further opportunities for power saving.

4. POWER MANAGEMENT POLICIES

Sections 2 and 3 discussed how to measure or estimate (Section 2) and how to adjust (Section 3) power usage. These components are highly hardware-specific and required in order to increase energy efficiency. However, they just provide the data and actions required to save energy – they do not

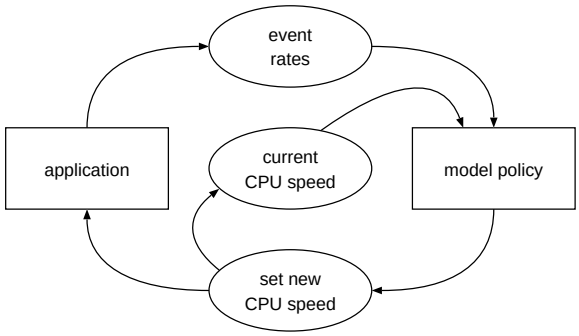

Figure 4: A simple control loop implementing a power management policy. From [13].

save any power on their own. Another component that uses the input data to select the power saving configuration providing the highest energy efficiency is needed – the so-called *power management policy*.

This article previously discussed the use of event counters to classify the workload (see Section 2.1). These counters span a multidimensional space when used as inputs for the power management policy. For each point in this space, the optimal power saving configuration can be pre-computed using a series of benchmarks (see Section 2) using constrained or non-constrained optimization. Different objective functions and constraints lead to different policies, some of which perform better than others.

Figure 4 shows a data flow graph where measurement (event rates, current CPU speed), adjustment (set new CPU speed), and policy (model policy) can be seen.

4.1 The Maximum-Degradation Policy

The so-called *maximum-degradation policy* as suggested by Weißel et al. [13] is represented by the optimization problem

$$\begin{aligned} &\text{minimize} && P \\ &\text{subject to} && T \leq p^{-1} \cdot T_{\text{opt}} \end{aligned} \tag{2}$$

where P is the power consumption, T is the duration of the task[2] with power saving and T_{opt} is the shortest duration the task would take without power saving. p is an adjustable parameter limiting the maximum performance degradation acceptable in order to save power. Values of $p = 0.9$ have been shown to work well [13].

4.2 The Generalized Energy-Delay Policy

A different approach called *generalized energy-delay policy* [11] is given by

$$\text{minimize} \quad P^{1-\alpha} \cdot T^{1+\alpha} \tag{3}$$

with P and T as defined in Equation 2 and $\alpha \in [-1; 1]$ being a variable. Modification of α transforms this policy into a number of policies previously suggested in literature:

- $\alpha = 1$ yields

$$\text{minimize} \quad T, \tag{4}$$

  which minimizes the execution time and thus maximizes the performance. Using this policy will implement a power saving mechanism that will always use the highest available frequencies to finish the task as quickly as possible. In systems that are not completely utilized, the processor can be put to sleep for a longer duration compared to running tasks at lower frequencies. This is called the *race-to-halt* approach.

- $\alpha = 0$ minimizes the energy usage:

$$\text{minimize} \quad P \cdot T =: E \tag{5}$$

- $\alpha = -1$ simplifies the optimization problem to

$$\text{minimize} \quad P, \tag{6}$$

  which minimizes the power consumption.

Values for α between zero and one will throttle tasks depending on their behavior: memory-bound processes achieve higher energy savings while sacrificing little power and can thus be run at a lower frequency than CPU-bound tasks. Although setting α to a value in $[-1; 0]$ will still throttle threads depending on the workload, the energy consumption will be higher than at $\alpha = 0$. These values might be useful in environments where power consumption needs to be reduced regardless of the energy used for the complete task (e.g., because the power consumption should not exceed a maximum value).

4.3 Adjustable Policies

Both the maximum-degradation policy and the generalized energy-delay policy are parametrized. Using this parameter, the operating system or the end user can adjust the policy to their needs. Requirements for power management depend on the purpose of the machine and might change over time: servers have different requirements than battery-powered devices, but servers can quickly turn into battery-powered devices over time, e.g., due to a power failure. Keeping the policy parameters adjustable at runtime allows the operating system to react on changed requirements and integrate with other components relevant for power saving.

4.4 Power Management Challenges

Whether energy can be saved by a certain decision depends on a wide range of factors. Modeling and predicting all of them as closely as possible is a complicated task. Some of the problems in generating accurate models of power consumption are outlined in this section.

4.4.1 Workload Prediction

As shown in Figures 1 and 2 the actual power consumption depends on the workload characteristics. Since we can only sample the behavior with a limited number of event counters, the precision of the prediction might suffer. Matching the behavioral patterns to the corresponding ideal power saving setting is error-prone, too: Figure 5 shows two pathological cases that require a good analysis of the task's behavior – the gzip benchmark is most efficient at the highest frequency, whereas the swim graph has its optimal point at a low (but not the lowest) frequency. It is obvious from this graph that energy saving decisions without insight into the type of workload are impossible.

[2] which is equal to the inverse performance

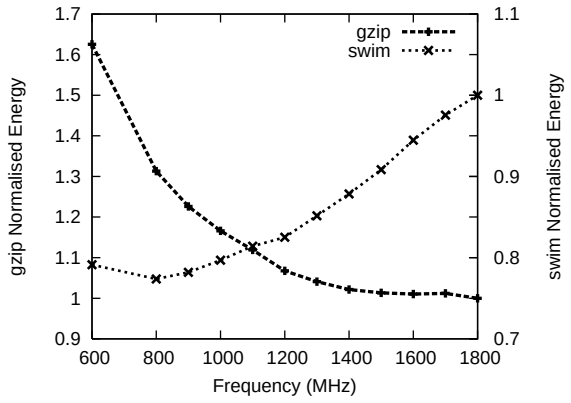

Figure 5: Normalized energy consumption of two benchmarks. From [11].

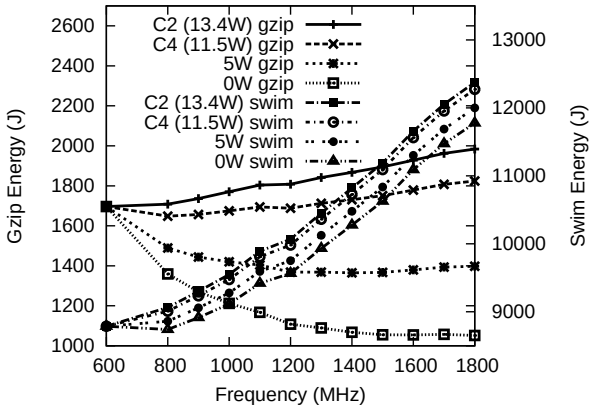

Figure 6: Energy consumption of two benchmarks when using race-to-sleep for the idle states C2, C4 and two hypothetical idle states with 5 and 0 W. From [11].

4.4.2 *Multiple Variables and Variable Memory System Performance*

Besides the CPU clock frequency, other frequencies in the system, such as the bus and memory frequency, might be adjustable at runtime. Such systems introduce another variable to be tested when finding the best power saving configuration, which quickly leads to an exploding number of configurations to be benchmarked ahead of time.

Snowdon et al. also found that memory performance is not completely unrelated to CPU clock speed. Some processor features like out of order execution or pre-fetching might become less effective at lower core frequencies. Those effects are hard to predict, because they are highly hardware-specific and few to no performance counters are available to measure them. Memory energy consumption additionally depends on the memory configuration (e.g., in single vs. multi channel configurations).

4.4.3 *Sleep States*

Tasks run at a lower frequency generally take longer to execute. The time delta between running a task at a high and at a low frequency can not be used to put the CPU to a sleep state if the system is otherwise idle. While this does not matter for heavily loaded machines, it is important for general purpose systems, such as laptops or tablets. Whether or not it is advisable to run a CPU-bound task at a high clock frequency, depends on the time delta and the power savings in sleep states the operating system might be able to put the processor into after task completion.

Consider a task starting at $t_0 = 0$ in two different clock frequencies f_{slow} and f_{fast} with a power consumption of P_{slow} and P_{fast}, respectively. We assume the task needs c cycles; at a frequency f_x the task will finish in

$$t_x = \frac{c}{f_x}, \qquad x \in \{\text{slow}, \text{fast}\}. \tag{7}$$

Without loss of generality, we can assume that $f_{\text{slow}} < f_{\text{fast}}$ and thus $t_{\text{slow}} > t_{\text{fast}}$, since $c = \text{const}$. The energy used then is

$$E_x = t_x \cdot P_x, \qquad x \in \{\text{slow}, \text{fast}\}. \tag{8}$$

However, the processor does not drop to zero energy usage after finishing the task – to get accurate results, we need to consider the energy used in the time $t_{\text{slow}} - t_{\text{fast}}$ where the processor is idle. Let us assume P_{idle} is the power consumption when the processor is idle. This leads us to a decision rule to find out whether we should run a task at a high frequency:

$$E_{\text{fast}} + (t_{\text{slow}} - t_{\text{fast}}) \cdot P_{\text{idle}} \leq E_{\text{slow}}. \tag{9}$$

Equation 9 shows that power consumption in idle mode can not be neglected when trying to minimize the energy consumption in interactive systems. We face a trade-off between the race-to-halt approach and running at a lower frequency for a longer time. The gzip benchmark in Figure 6 shows that with the improvement of idle states in processors, race-to-halt might be a better choice compared to running at low frequencies.

4.4.4 *Power-supply Efficiency and Temperature*

When employing dynamic voltage scaling, the power supply might not work equally efficient at different voltage levels. This imposes another difficulty on operating systems trying to save power, because reducing the voltage used to drive the CPU might not reduce the power consumption from battery or wall outlet either. Similarly, the CPU core temperature plays a role in the system's power consumption because of the power needed for fans and the higher leakage current caused by higher core temperatures.

4.4.5 *Switching Overhead*

Both frequency and voltage switching cause the CPU to be unavailable while the respective setting is scaled. This time is overhead, because energy is still being used, but no instructions are executed. Excessive switching of frequencies or voltages might be worse than running at a suboptimal performance setting for a short period of time. Accurate prediction of the duration of the currently running task is required to decide whether the switching overhead will be amortized by subsequent savings. Lu et al. call the decision boundary for this case the *break-even time* [7]. Snowdon et al. propose penalizing a frequency switch compared to staying at the current frequency [11].

5. NOTES ON IMPLEMENTATION

Since the behavior of a single process or thread is easier to predict than the workload of the whole operating system (assuming temporal locality), analysis and prediction is usually implemented per-process. This implies modifications of the dispatching mechanism. Because task switching is implemented in software (either because the hardware does not support task switching or because hardware task switching is not used), event counters will not be reset on task switches. As a consequence, implementations need to read the current values from the configured counters during the task switch. Storing the value enables measuring the event count per task by calculating the difference between the values at dispatch and preemption. Depending on the requirements of the implemented policies and storage space available, different methods to store the event counters per process come to mind: How much history should be kept per task, if any? Are all event counts saved, or only the significant ones? Are the last few bits relevant for the power management decision, or can they be omitted?

Activities of the operating system are not exempt from event counting. This leads to the finding that task switch and performance estimation code is attributed to either process currently being switched. Depending on the overhead associated with scheduling decisions, reading the event counters twice might lead to less distorted results. Reading those machine-specific registers might come with a non-negligible overhead that has to be considered, though. Interrupts will also be attributed to the currently running task. Unlike Weißel et al. who seem to ignore this completely, Snowdon et al. later show that the effect does not have a role in the measurements for all systems they considered [11].

When implementing power management policies, the time and complexity involved in solving the optimization problem typically associated with a policy has to be considered. The overhead can be reduced using pre-computed lookup tables or mathematical reformulations, e.g., to avoid floating point operations. For example, Equation 3 can be reformulated as

$$\text{minimize} \quad (1-\alpha)\log P + (1+\alpha)\log T. \tag{10}$$

6. CONCLUSION

To reduce the energy usage and increase energy efficiency, operating systems need to be able to measure or estimate current power consumption, predict a tasks workload and control a series of power saving mechanisms. The component that decides which measures to activate in order to save power is called a *power management policy*. Due to the complexity involved in accurately estimating and predicting power consumption, today's approaches are heuristic.

Due to lots of hardware-specific settings and sensors, power management benefits from adjustment to the hardware at hand. However, this requires significant effort by users or hardware manufacturers. To do this efficiently, vendors need to provide the hardware and have to be able to modify the operating system. We see a raising number of devices, especially in the mobile market, where this is the case.

From a software developer's point of view, a task should be as homogeneous as possible to simplify workload prediction in power management policies. Measured in energy efficiency, it might be better to use a thread for a single type of workload rather than using thread pooling for different tasks.

7. REFERENCES

[1] Frank Bellosa. *The Case for Event-Driven Energy Accounting*. Tech. rep. Erlangen: Friedrich Alexander Universität, 2001.

[2] Hewlett-Packard Corporation et al. *Advanced Configuration and Power Interface Specification, Revision 5.0*. Dec. 6, 2011.

[3] Timo Hönig, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. "Extending Mobile Devices by Exploiting Remote Resources". In: *Proceedings of ACM European Conference on Computer Systems (EuroSys 2010), Poster Session*. Ed. by ACM SIGOPS. Paris, France, 2010.

[4] Timo Hönig, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. "ProSEEP: A Proactive Approach to Energy-Aware Programming". In: *Proceedings of the 2012 USENIX Annual Technical Conference (ATC 2012), Poster Session*. Ed. by USENIX Association. Boston, MA, USA, 2012.

[5] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*. 253669-046US. 2013.

[6] ARM Ltd. *Performance Monitor Unit example code for ARM11 and Cortex-A/R*. May 2, 2013. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka4237.html.

[7] Yung-Hsiang Lu and Giovanni De Micheli. "Comparing system level power management policies". In: *Design & Test of Computers, IEEE* 18.2 (2001), pp. 10–19.

[8] Akihiko Miyoshi et al. "Critical power slope: understanding the runtime effects of frequency scaling". In: *Proceedings of the 16th international conference on Supercomputing*. ACM. 2002, pp. 35–44.

[9] Trevor Pering and Robert Brodersen. "Energy efficient voltage scheduling for real-time operating systems". In: *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS'98, Work in Progress Session*. 1998.

[10] David C Snowdon, Sergio Ruocco, and Gernot Heiser. "Power management and dynamic voltage scaling: Myths and facts". In: *Proceedings of the 2005 Workshop on Power Aware Real-time Computing, New Jersey, USA*. 2005.

[11] David C. Snowdon et al. "Koala: a platform for OS-level power management". In: *Proceedings of the 4th ACM European conference on Computer systems*. EuroSys '09. Nuremberg, Germany: ACM, 2009, pp. 289–302.

[12] Intel Corp. Taylor Kidd. *(update) C-states, C-states and even more C-states*. Mar. 27, 2008. URL: http://software.intel.com/en-us/blogs/2008/03/27/update-c-states-c-states-and-even-more-c-states/.

[13] Andreas Weißel and Frank Bellosa. "Process cruise control: event-driven clock scaling for dynamic power management". In: *Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*. ACM. 2002, pp. 238–246.

Profiling von Software-Energieverbrauch

Michael Fiedler
Friedrich-Alexander-Universität Erlangen-Nürnberg
michael.fiedler@informatik.stud.uni-erlangen.de

KURZZUSAMMENFASSUNG

Energieverbrauch ist ein wichtiger Aspekt beim Entwurf heutiger Hard- und Software. Gerade im Bereich von mobilen Geräten, bei denen die Laufzeit durch die Batteriekapazität stark beschränkt ist, gleichzeitig aber Ansprüche an die Rechenleistung und Datenübertragung zunehmen, kommt einer möglichst geringen Leistungsaufnahme eine wesentliche Rolle zu. Das Profiling des Energieverbrauchs von Software, also die Zuordnung von Energieverbrauch zu denjenigen Programmstrukturen, die auf seine Höhe Einfluss haben, kann Entwickler auf die energiehungrigen Problemstellen seiner Programme hinweisen, sodass diese gezielt verbessert werden können. Diese Arbeit gibt hierzu einen Einblick in Konzepte und Probleme aus dem Bereich des Profilings von Software-Energieverbrauch. Zusätzlich werden mehrere Umsetzungen von Profilern genauer betrachtet.

1. EINLEITUNG

Bei heutigen Rechnern spielt neben Geschwindigkeit auch der Energieverbrauch eine wichtige Rolle. Dies ist zum Einen durch eine im Alltagsleben zu beobachtende zunehmende Verbreitung von mobilen Geräten wie Notebooks, Smartphones und Tablet-Computern bedingt, bei denen die Gerätelaufzeit durch die verfügbare Batteriekapazität beschränkt ist. Auch bei batteriebetriebenen Sensorknoten in Sensornetzwerken [1] existiert diese Problematik. Darüber hinaus ist ein möglichst geringer Energieverbrauch auch bei größeren Rechnern oder Rechenanlagen ein Thema: Hier sind die Senkung von Stromkosten, der Umweltschutz (beispielsweise aus Marketing-, Image- oder rechtlichen Gründen) und das Abführen der durch den Energieverbrauch entstehenden Wärme anzuführen. Nicht nur die Hardware, sondern auch die Software trägt maßgeblich dazu bei, dem Ziel möglichst energieeffizienter Systeme näher zu kommen – gerade auf der Software-Ebene besteht ein besserer Überblick über das Gesamtsystem, der für Energieverbrauchsoptimierungen genutzt werden kann.

Um energieeffiziente Software zu bauen, benötigen Entwickler geeignete Werkzeuge, die sie bei ihrer Aufgabe unterstützen. Diese sollen diejenigen Stellen im System mit Verbesserungsmöglichkeiten bezüglich des Energieverbrauchs aufzeigen, gegebenenfalls vorhandene Fehler in der Energieverwaltung (zum Beispiel sogenannte *Wake-Lock-Fehler* in Smartphone-Anwendungen [2]) entdecken und nach Änderungen am Quelltext die Auswirkungen auf den Energieverbrauch beziffern. Dafür ist es notwendig, den Energieverbrauch anteilig den ihn verursachenden Hard- und Software-Komponenten auf verschiedenen Genauigkeitsebenen (Prozedur/Funktion, Prozess, einzelnes Gerät, Teilsystem des Betriebssystem, ...) zuordnen zu können – nicht nur, um ein bereits umgesetztes System bei seiner Ausführung zu beobachten, sondern auch, um noch nicht im Einsatz befindliche Systeme wie einzelne Sensorknoten in ihrem Energieverbrauchsverhalten und damit in ihrer Laufzeit abschätzen zu können [1].

Der beschriebenen Aufgabe widmet sich das Profiling von Software-Energieverbrauch mit der Leitfrage "An welcher Stelle meiner Software wird wie viel (qualitativ/quantitativ) Energie verbraucht?". Während im Bereich des Profilings für die Rechenleistung des Prozessors schon seit vielen Jahren Programme wie Gprof [3] etabliert sind, ist dies beim Energieverbrauchsverhalten noch nicht der Fall. Diese Ausarbeitung soll daher einen Einblick in Problemstellungen und Lösungsansätze aus Veröffentlichungen zum Thema "Profiling von Software-Energieverbrauch" bieten.

Die allgemeine Vorgehensweise beim Profiling von Software-Energieverbrauch besteht dabei aus folgende Schritten:

1. *Durchlaufen des Programms*: Dies kann durch Ausführen des untersuchten Programms in einer für Energiemessungen instrumentierten realen [4] oder simulierten [1] Umgebung durchgeführt werden. Auch die zusätzliche Verwendung symbolischer Ausführung [5] für eine maximale Programmpfadüberdeckung ist dabei möglich [6].

2. *Ermitteln des bei Ausführung benötigten Energieverbrauchs*: Dies kann beispielsweise durch eine tatsächliche Messung mit Hilfe eines Multimeters [4], durch Schätzung basierend auf einem Modell [1] oder unter Zuhilfenahme einer Ersatzmetrik und eines für diese ermittelten Hardware-spezifischen Energieverbrauchsprofils [6] geschehen.

3. *Herstellen der Verbindung zwischen dem aktuellen Energieverbrauch und den Programmstrukturen, die darauf Einfluss haben*: Eine Möglichkeit dafür ist die Analyse des Stapelspeichers während der Programmausführung [4, 7].

Als problematisch beim dritten Punkt erweist sich dabei der *asynchrone Energieverbrauch* [7]. Im Gegensatz zum synchronen Fall, bei dem der Energieverbrauch zeitgleich mit der Ausführung des ihn verursachenden Programmbestandteils auftritt (zum Beispiel beim Energieverbrauch durch das Ausführen von Maschinenbefehlen durch den Prozessor), tritt asynchroner Energieverbrauch verzögert zum

Verursachungszeitpunkt auf (zum Beispiel bei Ein- und Ausgabeoperationen mit Pufferspeichern, wo der eigentliche Gerätezugriff erst nach mehreren Ein- und Ausgabeoperationen stattfindet). Auch dieser zeitlich verzögerte Energieverbrauch muss also im Profiling berücksichtigt werden, um korrekte Ergebnisse zu erhalten. Insbesondere ist er bei der Betrachtung von Anwendungssoftware für mobile Geräte zu beachten, wo asynchroner Energieverbrauch eine wichtige Rolle spielt [2].

Diese Arbeit ist wie folgt aufgebaut: In Abschnitt 2 werden Entwurfsaspekte von Software-Energieverbrauch-Profilers diskutiert und eine Übersicht über Problemstellungen und mögliche Lösungsansätze gegeben. Danach wird in Abschnitt 3 auf mehrere Arbeiten zum Profiling genauer eingegangen. Abschließend folgt eine Zusammenfassung der wichtigsten Aspekte dieser Arbeit.

2. ENTWURFSASPEKTE FÜR PROFILER

In diesem Abschnitt werden diverse Konzepte betrachtet, mit deren Hilfe Profiler für Software-Energieverbrauch in der Forschung der letzten Jahre umgesetzt wurden. So unterscheiden sich die Arbeiten unter Anderem in folgender Hinsicht:

- *Messung des Energieverbrauchs*: Die Messung kann zum Beispiel per Hardware-Instrumentierung oder per Schätzung mit Hilfe von Ersatzmetriken erfolgen.
- *Granularität*: Die Zuordnung des Energieverbrauchs kann unter Anderem auf der Ebene von Prozessen oder von Funktionen stattfinden.
- *Grad der Programmpfadabdeckung*: Wird bei der Erstellung des Energieverbrauchsprofils nur ein konkret ausgeführter Programmpfad berücksichtigt oder wird angestrebt, die Zahl der überprüften möglichen Pfade zu maximieren?
- *Vorgehensweise zur Verknüpfung von Enerieverbrauch und Programmstrukturen*
- *asynchrones Energieverbrauchsverhalten*: Wird die Energie nur der jeweils in Ausführung befindlichen Programmstelle zugeordnet, oder werden verzögerte Geräteaufrufe (zum Beispiel bedingt durch Pufferspeichereffekte) berücksichtigt? Und wie wird die Zuordnung umgesetzt?

2.1 Messung des Energieverbrauchs

Um eine korrekte Zuordnung zwischen dem Energieverbrauch bei der Ausführung einer Software und den ihn verursachenden Programmstrukturen herstellen zu können, ist eine ständige Verfolgung des Energieverbrauchs notwendig. Da heutige Rechner normalerweise nicht in hinreichendem Maße mit Sensoren ausgestattet sind, über die per Software die aktuellen Energieverbrauchswerte der einzelnen Geräte ausgelesen werden können, sind alternative Methoden erforderlich.

Eine Messung kann daher entweder durch direkte Instrumentierung des Rechners geschehen, auf dem das Programm ausgeführt wird, oder aber man bedient sich eines Modells bzw. einer Ersatzmetrik, über die der reale Energieverbrauch indirekt abgeschätzt wird. Der Vorteil der letzteren Variante ist das Wegfallen des Aufwands dafür, die physische Messinfrastruktur bei der Ausführung der untersuchten Software anzubringen. So können Geräte auch emuliert werden oder das Profiling erfolgt während der Programmausführung gleich auf dem untersuchten Rechner.

2.1.1 Direkte Instrumentierung der Hardware mit Messgeräten

Flinn et al. verwenden für ihr Profiling-Werkzeug PowerScope [4] ein Digitalmultimeter, mit dem sie den Stromverbrauch des gesamten untersuchten Systems verfolgen. Die durch Abtastung erzeugten Messwerte werden auf einem separaten Rechner gesammelt und später mit den zeitgleich erhobenen Daten über die aktuell ausgeführte Programmstelle verbunden (siehe dazu auch Abbildung 4 unten). Ein externes Messgerät wurde hierbei einem in den untersuchten Rechner eingebauten vorgezogen, um eine Beeinflussung der Messergebnisse durch die Messung selbst zu verhindern.

2.1.2 Modellierung des Energieverbrauchs der einzelnen Rechnerbestandteile

Als Alternative zu einem Profiling mit direkter Instrumentierung des verwendeten Rechners kann der Energieverbrauch auch über Energieverbrauchsmodelle geschätzt werden [1, 8, 2, 7]. Dabei wird zum Beispiel die Leistungsaufnahme verschiedener Zustände eines Geräts ermittelt. Bei der Ausführung des untersuchten Programms wird dann von den aktuellen Zuständen aller Geräte des Systems auf den momentanen Energieverbrauch geschlossen. Vor ihrer Verwendung können diese Modelle auf ihre Anwendbarkeit, Korrektheit und Genauigkeit hin überprüft werden – hier wiederum mit Hilfe von realen Stromverbrauchsmessungen an einer Referenzhardware. Für das eigentliche Software-Profiling wird dann das validierte Energieverbrauchsmodell angewandt.

In der Literatur finden sich verschiedene Vorgehensweisen, darunter die folgenden:

- *Modell für die Emulation von Hardware und Ausführungsumgebung*: Bei AEON [1] wird für die Analyse eines Knotens in einem Sensornetzwerk die in den Sensorknoten verwendete Hardware emuliert. Zunächst wird ein Modell für den Energieverbrauch der einzelnen Geräte in ihren verschiedenen Gerätezuständen erstellt. Anschließend wird die untersuchte Software mit Hilfe eines Emulators der Hardware ausgeführt. Dabei kann über die Zustandswechsel der Geräte und die im Modell enthaltene Leistungsaufnahme der einzelnen Geräte in den jeweiligen Gerätezuständen der Energieverbrauch des Gesamtsystems ermittelt werden.
- *Modell für das Nachverfolgen von Systemaufrufen* (engl. *system call tracing*): Bei Eprof von Pathak et al. [8, 2] hingegen wird ein Modell verwendet, das die bei der Programmausführung auftretenden Systemaufrufe betrachtet. Da über die Systemaufrufe zum Beispiel Ein- und Ausgabeoperationen abgehandelt werden, werden die Systemaufrufe als Auslöser für Zustandsübergänge der Geräte betrachtet. Über die im Energieverbrauchsmodell enthaltenen Verbrauchswerte der Gerätezustände wird ein Wert für den Gesamtenergieverbrauch ermittelt.

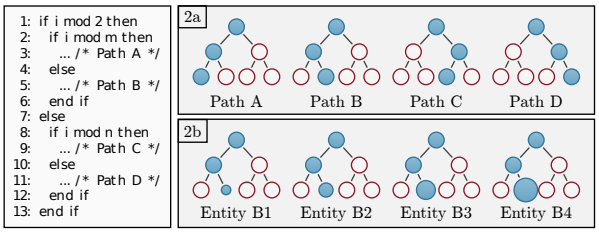

Abbildung 1: Programmpfade und Programmeinheiten bei SEEP. Die Größe der blauen Knoten der Graphen stellt den Energieverbrauch des jeweiligen Programmabschnitts schematisch dar. Quelle: [6]

2.2 Granularität der Zuordnung des Energieverbrauchs zu Programmstrukturen

Um Software-Programmierern einen möglichst guten Überblick über energieaufwändige Bereiche ihres Programms zu ermöglichen, betrachten die in dieser Arbeit vorgestellten Profiling-Konzepte die Programmstruktur-Energieverbrauch-Zuordnung auf der Ebene von Funktionen (und auch Prozessen, bei [2] werden zusätzlich Threads und Systemaufrufe genannt). Auch auf höheren Abstraktionsebenen wie bei der Betrachtung von ganzen Rechnern, Sensornetzwerken [9] oder im Cloud-Computing-Bereich [10] existieren Ansätze zum Profiling. Auf diese wird jedoch in dieser Arbeit nicht weiter eingegangen.

2.3 Programmpfadabdeckung

Die Ausführung von Software mit bestimmten Aufrufparametern und unter bestimmten Umgebungsbedingungen führt zu einem Programmdurchlauf, der nur einen von vielen möglichen darstellt (siehe dazu Abbildung 1). Der resultierende Ausführungspfad durch das Programm muss dann jedoch nicht repräsentativ für das Energieverbrauchsverhalten weiterer möglicher Ausführungspfade sein. Hinzu kommt, dass selbst innerhalb eines Ausführungspfads das Energieverbrauchsverhalten von der konkreten Eingabe und Umgebung abhängt. Das durch einen einzelnen Profiling-Durchlauf gelieferte Energieverbrauchsprofil kann also nur bedingt Aussagen über das Gesamtverhalten der untersuchten Software liefern. Idealerweise sollte das Profil also einen möglichst großen und repräsentativen Anteil der Wege durch ein Programm berücksichtigen.

Eine Möglichkeit, die Überdeckung der Ausführungspfade zu erhöhen, ist die Zuhilfenahme symbolischer Ausführung [5] von Programmen. Mit Hilfe dieser Technik werden bei SEEP [6] Eingabewerte für eine möglichst große Programmpfadüberdeckung gefunden. Anschließend werden für jeden Programmpfad für mehrere Eingabedaten Programmeinheiten erstellt. Diese werden dann mit dem Profiler untersucht und die Ergebnisse können für den Programmpfad zusammengefasst werden.

2.4 Verknüpfung von Energieverbrauch und Programmstrukturen

Um Energieverbrauchsprofile für Software zu erstellen, ist es erforderlich, den Zusammenhang zwischen dem Maschinencode des untersuchten Programms, der im Prozessor ausgeführt wird, und der zugehörigen Programmstruktur auf Programmiersprachenebene herzustellen. Erst dann kann der während der Ausführung der Maschinenbefehle festgestellte Energieverbrauch zu denjenigen Programmbestandteilen wie einzelnen Funktionen zugeordnet werden, die für den Programmierer interessant sind. Die Zuordnung von Programmzähler zur zugehörigen Funktion ist hierbei über die Symboltabelle der ausgeführten Programmdatei bzw. der verwendeten Programmbibliotheken möglich. Dazu finden sich folgende zwei Möglichkeiten, um vom Programmzähler über die Symboltabelle der ausgeführten Programmdateien bzw. der verwendeten Programmbibliotheken an die zugehörigen:

- *Abtasten der Werte des Programmzählers*: Bei PowerScope [4] wird regelmäßig der aktuelle Wert des Programmzählers des Prozessors erfasst, sodass er später über die Symboltabelle des ausgeführten Programms zu einer Funktion im Programmquelltext zugeordnet werden kann.
- *Nachverfolgen der Funktionsaufrufe*: Eprof von Schubert et al. [7] und Eprof von Pathak et al. [8, 2] betrachten die Programmaufruffolge auf dem Stapelspeicher, bei AEON [1] wird jeder Funktionsaufruf registriert.

2.5 Asynchrones Energieverbrauchsverhalten

Bei der Zuordnung von gemessenem Energieverbrauch zu Software-Programmstrukturen sind zwei verschiedene Fälle für den zeitlichen Zusammenhang zwischen der Ausführung eines Programmbefehls bzw. einer Programmstruktur und dem daraus resultierenden Energieverbrauch zu unterscheiden:

1. Der Energieverbrauch findet zeitgleich mit der Ausführung statt (*synchroner Energieverbrauch*).
2. Der Energieverbrauch findet zeitlich verzögert zur Ausführung oder auf einen längeren Zeitraum verteilt statt (*asynchroner Energieverbrauch*).

Im synchronen Fall kann der Energieverbrauch einfach mit den zum Messzeitpunkt ausgeführten Maschinenbefehl und damit der zugehörigen Programmeinheit verknüpft werden – es müssen also zum Beispiel nur der Energieverbrauch und der Programmzähler oder die Reihenfolge der Funktionsaufrufe auf dem Stapelspeicher gleichzeitig abgegriffen werden. Der asynchrone Fall ist komplizierter zu handhaben. Im Folgenden wird zunächst genauer betrachtet, in welchen Fällen Asynchronität auftritt [2]. Anschließend wird auf mögliche Vorgehensweisen eingegangen, mit deren Hilfe zeitverzögert auftretender Energieverbrauch zu Programmstrukturen zugewiesen werden kann [2].

2.5.1 *Ursachen für und Problemstellungen bei asynchronem Energieverbrauchsverhalten*

Asynchrones Energieverbrauchsverhalten kann verschiedene Ursachen haben, auf die in diesem Abschnitt eingegangen wird. Neben den verschiedenen Betriebszuständen von Geräten und das Beibehalten eines relativ viel Energie benötigenden Aktivitätsbereitschaftszustands nach einem Gerätezugriff sind gerade in heutigen Mobiltelefonen auch Wachzustandssperren (engl. *wake locks*) sowie Spezialgeräte wie GPS-Empfänger von Bedeutung.

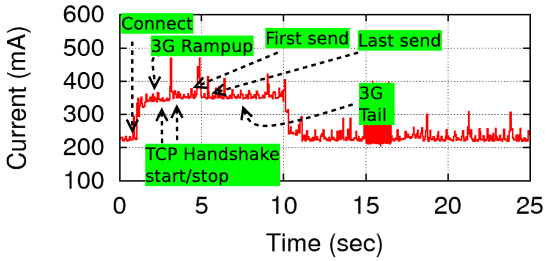

Abbildung 2: Stromverbrauch eines Mobiltelefons beim Senden von 5 Dateneinheiten zu je 10 kB über das Mobilfunknetz (3G) mit den *Sendebefehlen direkt nach dem Verbindungsaufbau.* Quelle: [2]

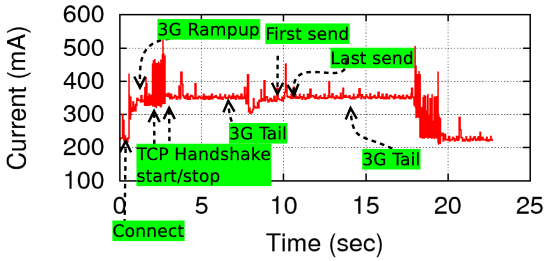

Abbildung 3: Stromverbrauch eines Mobiltelefons beim Senden von 5 Dateneinheiten zu je 10 kB über das Mobilfunknetz (3G) mit den *Sendebefehlen 5 Sekunden nach dem Verbindungsaufbau.* Quelle: [2]

Die in heutigen Rechnern verbauten Geräte haben verschiedene Betriebszustände, die dabei helfen, ihren Energieverbrauch gering zu halten. Neben einem Grundzustand, in dem keine bzw. möglichst wenig Energie verbraucht wird, kann es mehrere Zustände für Aktivitätsphasen geben [2]. Darunter kann zum Beispiel bei Netzwerkgeräten wie WLAN- und Mobilfunkgeräten ein "Schweifzustand" (engl. *tail state*) sein, von dem aus schnell in einen Sendezustand gewechselt werden kann, aber in dem im Vergleich zum Sendezustand weniger Energie verbraucht wird. In diesem Schweifzustand kann jedoch deutlich mehr Energie als im Grundzustand benötigt werden. Wird auf das Gerät zugegriffen, so muss vom aktuellen Gerätezustand in den Sendezustand gewechselt werden. Anschließend geht das Gerät in den Schweifzustand über und verweilt dort eine Zeit lang.

Der Schweifzustand erweist sich für die Zuordnung des aktuellen Energieverbrauchs zu denjenigen Programmeinheiten, die ihn verursachen, als Problem. Im Gegensatz Geräten, die ein synchrones Energieverbrauchsverhalten aufweisen, wie Prozessoren bei der Ausführung von Maschinenbefehlen, wirkt die Geräteaktivität durch das Verweilen im Energie verbrauchenden Schweifzustand noch nach: Dessen Ursache hat nichts mit der gerade ausgeführten Programmanweisung zu tun, sondern mit einem Gerätezugriff aus einer anderen Programmanweisung, die irgendwann in der Vergangenheit ausgeführt wurde.

Eine Illustration der Problematik findet sich in den Abbildungen 2 und 3. Sie zeigen den Verlauf des Stromverbrauchs eines Mobiltelefons über das Mobilfunknetzwerk (3G) beim Übertragen von fünf Dateneinheiten zu je 10 kB mittels fünf Sendebefehlen. Nach dem Aufbau einer TCP-Verbindung werden die Daten gesendet. Im ersten Fall (Abbildung 2) folgt der Sendebefehl für die Datenpakte zeitlich direkt auf den Verbindungsaufbau. Im Anschluss an den letzten Sendebefehl ist zu beobachten, dass über mehrere Sekunden hinweg der Stromverbrauch auf hohem Niveau verweilt (Schweifzustands des Mobilfunkgeräts). Erst dann kehrt er mit einem Sprung wieder auf sein Ausgangsniveau zurück. Gibt es nun zusätzlich eine Verzögerung zwischen dem Verbindungsaufbau und dem Senden der Datenpakete (Abbildung 3), so verlängert sich die Zeit im Schweifzustand, in dem ein im Vergleich zum Grundzustand erhöhter Stromverbrauch auftritt, zusätzlich. Um den Stromverbrauch auf einzelne Programmbestandteile aufzuteilen, ist daher eine passende Strategie erforderlich, in die die Zustandsübergänge der einzelnen Geräte einfließen und mit deren Hilfe die verbrauchte Energie auf die beteiligten Funktionsaufrufe aufgeteilt wird. Auf mögliche Vorgehensweisen wird in Abschnitt 2.5.2 weiter eingegangen.

Ähnliche Effekte wie der Schweifzustand von Geräten können auch in anderen Fällen auftreten, in denen Geräte nicht unmittelbar nach dem Ende ihrer Aktivität auf den energiesparsamsten Zustand wechseln [2]. In aktuellen Betriebssystemen für mobile Geräte gibt es etwa Programmierschnittstellen, die es Anwendungsentwicklern ermöglichen, eine Wachzustandssperre (engl. *wake locks*) zu setzen, bis sie von ihnen explizit wieder gelöst wird. Profiler, die sich dieser Wachzustandssperren bewusst sind, können daher auch Fehler in der Benutzung dieser Programmierschnittstellen (engl. *wake lock bugs*) aufdecken. Darüber hinaus gibt es Geräte wie GPS-Empfänger und Kameras, die nach ihrer Aktivierung über einen längeren Zeitraum in einem aktiven, energieverbrauchenden Zustand verweilen können und in ähnlicher Weise von Profilern berücksichtigt werden sollten.

2.5.2 *Zuordnungsstrategien bei asynchronem Energieverbrauchsverhalten*

Pathak et al. [2] gehen auf mehrere Möglichkeiten ein, asynchronen Energieverbrauch zu den ihn auslösenden Programmeinheiten zuzuordnen:

1. gleichmäßige oder gewichtete Aufteilung auf alle Programmeinheiten
2. Zuweisung zum ersten oder letzten Programmbestandteil, der das Verweilen auf einem energieverbrauchenden Gerätezustand zur Folge gehabt hätte

Problematisch bei ersten Punkt erweist sich demnach zunächst das Festlegen und die Umsetzung der Gewichtung, die das resultierende Energieverbrauchsprofil schwerer verständlich machen. Zusätzlich kommt hinzu, dass dem Benutzer des Profilers durch solche Zuordnungen suggeriert wird, dass das Entfernen einer Programmeinheit mit einer starken Gewichtung im ersten Fall bzw. im zweiten Fall der Programmeinheit, die die gesamten asynchronen Energieverbrauch zugewiesen bekommen hat, den Energieverbrauch seiner Anwendung entsprechend absenkt. Entgegen dessen wird sich der Energieverbrauch dann jedoch zum Großteil auf andere Programmeinheiten verteilen. Um eine intuitive Verwendung des Energieverbrauchsprofilers zu ermöglichen, kommen Pathak et al. [2] daher zum Schluss, dass in jedem Fall ein expliziter Vermerk im Energieverbrauchsprofil

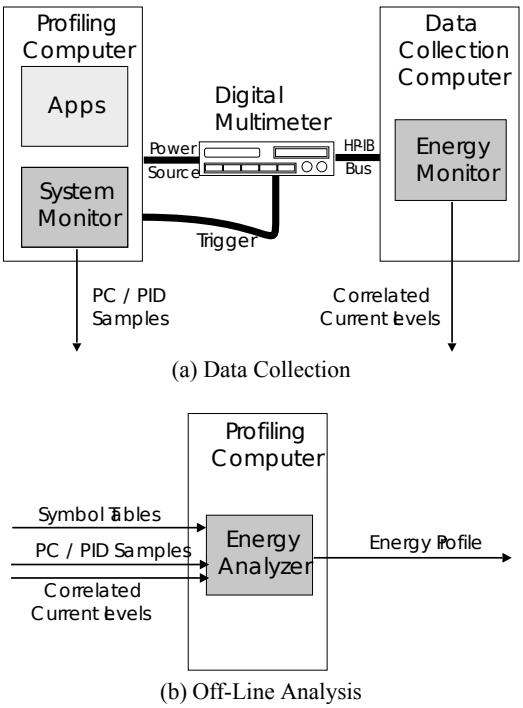

Abbildung 4: Überblick über PowerScope. (a) Abtastung von Werten für aktuellen Stromverbrauch, Programmzähler und Prozess-ID des untersuchten Rechners. (b) Zusammenführen der erhaltenen Werte mit der Symboltabelle des untersuchten Programms zum Energieverbrauchsprofil. Quelle: [4]

notwendig ist, der den Umgang mit asynchronem Energieverbrauch für den Benutzer ersichtlich macht, zum Beispiel eine Angabe des Energieverbrauchs eines Systemaufrufs mit einem Tupel (Energieverbrauch des eigentlichen Systemaufrufs, Energieverbrauch des nachfolgenden Schweifzustands).

3. BEISPIELE FÜR UMSETZUNGEN VON ENERGIEVERBRAUCH-PROFILERN

In den letzten anderthalb Jahrzehnten wurden Arbeiten aus verschiedenen Forschungsprojekten veröffentlicht, die Umsetzungen von Profilern für Software-Energieverbrauch zum Thema haben. In diesem Abschnitt werden drei der Projekte näher vorgestellt:

- *PowerScope* (Flinn et al., 1999) [4]: Profiling mittels Abtastung von Wertpaaren aus dem Programmzähler im Prozessor und einem Stromverbrauchsmesswert des untersuchten Rechners
- *AEON* (Landsiedel et al., 2005) [1]: Profiling mittels Energieverbrauchsmodellierung der Gerätezustände eines Sensorknotens und Ausführen auf simulierter Hardware
- *Eprof* (Pathak et al., 2011, 2012) [8, 2]: Profiling mittels Nachverfolgen von Funktions- und Systemaufrufen und Modellierung der Gerätezustände von Smartphones

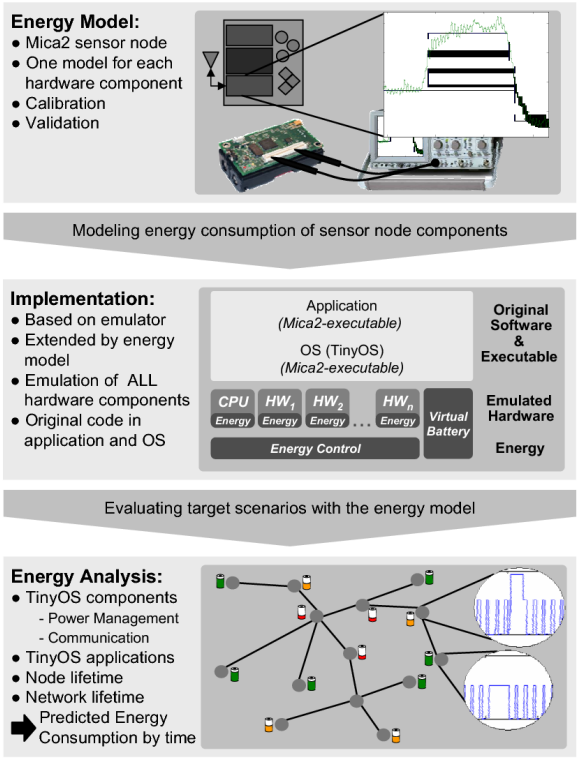

Abbildung 5: Überblick über AEON. Quelle: [1]

3.1 PowerScope

PowerScope [4] führt sein Profiling in zwei Schritten durch: In der ersten Phase (Abbildung 4a) wird eine Abtastung[1] von Wertpaaren bestehend aus dem aktuellen Programmzähler des Prozessors und der aktuellen Prozess-ID auf dem Rechner, auf dem das untersuchte Programm läuft (Profiling-Rechner), durchgeführt. Die Abtastung wird dabei durch ein digitales Multimeter ausgelöst, das gleichzeitig den aktuellen Stromverbrauch des gesamten Profiling-Rechners erfasst und an einen zweiten Rechner (Datensammlungsrechner) weiterleitet. Auf dem unteruchten Rechner läuft für das Erfassen der Werte ein angepasstes Betriebssystem, das zusätzlich für jeden Prozess den Pfad zur ausgeführten Programmdatei und bei jedem Laden von Programmbibliotheken auch deren Pfad erfasst.

In der zweiten Phase (Abbildung 4b) werden die erhaltenen Abtastwerte auf dem untersuchten Rechner mit den Symboltabellen der ausgeführten Programme und verwendeten Programmbibliotheken verknüpft und Energieverbrauchswerte aus den Strommesswerten und der Länge der Abtastintervalle berechnet. Als Ergebnis liefert PowerScope Energieverbrauchswerte für ganze Prozesse und Unterbrechungsroutinen, aufgeschlüsselt auf ihre verwendeten Funktionen.

[1]Die Abtastfrequenz kann vom Benutzer eingestellt werden, für Messungen in [4] werden Abtastintervalle von etwa 1,6 ms angegeben.

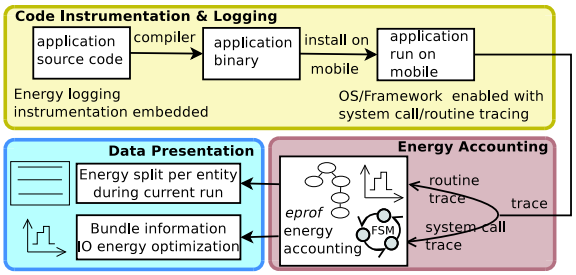

Abbildung 6: Überblick über die Architektur des Energieverbrauch-Profilers Eprof von Pathak et al. Quelle: [2]

3.2 AEON

Das AEON-Projekt [1] (siehe Abbildung 5) modelliert das Energieverbrauchsverhalten der Sensorknotenplattform Mica2 mit dem Betriebssystem TinyOS [11], um Aussagen über die Laufzeit von auf der Plattform ausgeführten Anwendungen treffen zu können. Zusätzlich implementiert das Projekt Profiling-Funktionalität. Bei AEON wird für jede Hardware-Komponente ein Modell für die verfügbaren Gerätezustände und den jeweiligen Energieverbrauch in diesen erstellt. Die Modelle werden mit Hilfe von Strommessungen kalibriert und validiert. Darauf basierend sind die Modelle im Sensorknotenemulator AVRORA [12] implementiert. Für das Profiling bzgl. des Energieverbrauchs der CPU werden die Funktionen im Programmquelltext auf die die Adressen im Objektcode abgebildet und während der Programmausführung alle Funktionsaufrufe mitgeschnitten.

3.3 Eprof (Pathak et al.)

Pathak et al. verfolgen für ihren Energieverbrauch-Profiler Eprof [8, 2] neben den verwendeten Funktionen zusätzlich nach, welche Systemaufrufe in den untersuchten Smartphone-Programmen aufgerufen werden (engl. *system call tracing*). Ihr Ziel ist es dabei, das Energieverbrauchsverhalten vollständiger abzubilden, als es durch eine alleinige direkte Verknüpfung von Gerätezuständen und der Gerätenutzung durch ein Programm möglich ist (sog. benutzungsbasierte Modellierung, engl. *utilization-based modelling*). So wird zum Beispiel das in Abschnitt 2.5 besprochene asynchrone Energieverbrauchsverhalten mit berücksichtigt.

Abbildung 6 zeigt das Vorgehen beim Profiling mit Eprof. Für das Android-Betriebssystem[2] werden die untersuchten Anwendungen, die verwendete Laufzeitumgebung und der Betriebssystemkern so präpariert, dass zur Laufzeit Funktions- und Systemaufrufe nachverfolgt werden können (Bereich *Code Instrumentation & Logging* in der Abbildung). Bei Anwendungen, deren Quelltext nicht verfügbar ist, werden dennoch die Aufrufe in der Laufzeitumgebung mit berücksichtigt. Basierend auf diesen erhaltenen Informationen wird in einem zweiten Schritt (Bereich *Energy Accounting* in der Abbildung) den ausgeführte Programmstrukturen, erhalten über die aufgerufenen Funktionen, ein Energieverbrauchswert zugeordnet. Der Energieverbrauch wird dabei unter Zuhilfenahme eines Modells für die verschiedenen Geräte in Form eines endlichen Automaten erhalten (die Zustandsübergänge des Automaten werden durch die verschiedenen Systemaufrufe ausgelöst). Diese erhaltenen Informationen werden dann dem Nutzer als Ergebnis dargestellt (Bereich *Data Presentation* in der Abbildung) – zum Einen als Energieverbrauch der einzelnen Programmbestandteile, zum Anderen als sog. *bundle*-Repräsentation für einen detaillierteren Einblick in die asynchron in den Geräten für Ein-/Ausgabe verbrauchte Energie.

[2]Eprof wurde neben Android auch für Windows Mobile umgesetzt.

4. ZUSAMMENFASSUNG

In dieser Arbeit wurde zunächst auf die grundlegenden Schritte beim Profiling von Software-Energieverbrauch eingegangen: Das untersuchte Programm wird durchlaufen und der dabei entstehende Energieverbrauch wird zu denjenigen Programmbestandteilen zugeordnet, die für ihn und seine Höhe ursächlich sind. Danach wurden verschiedene Aspekte, die es beim Entwurf eines Profilers für Software-Energieverbrauch zu beachten gilt, besprochen. Abschließend wurden mit PowerScope, AEON und Eprof drei Beispiele für Profiler-Umsetzungen genauer betrachtet.

Obwohl, wie dargelegt wurde, verschiedene Ansätze zum Profiling von Software-Energieverbrauch existieren, hat sich bisher keine Umsetzung für den Alltagsgebrauch in der Software-Entwicklung etabliert. Denkbare Aufgaben für weitere Arbeiten in diesem Bereich können daher, neben der zusätzlichen Betrachtung weiterer Hardware-Komponenten, in der Integration der verschiedenen Ansatzpunkte und verbesserter Anwenderfreundlichkeit liegen, um das sog. *energiegewahre Programmieren* (engl. *energ-aware programming*) [6] in der Praxis zu vereinfachen.

5. LITERATUR

[1] O. Landsiedel, K. Wehrle, and S. Gotz, "Accurate prediction of power consumption in sensor networks," in *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, EmNets '05, (Washington, DC, USA), pp. 37–44, IEEE Computer Society, 2005.

[2] 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 european conference on Computer Systems*, EuroSys '12, (New York, NY, USA), pp. 29–42, ACM, 2012.

[3] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," in *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, SIGPLAN '82, (New York, NY, USA), pp. 120–126, ACM, 1982.

[4] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pp. 2–10, 1999.

[5] C. Cadar and K. Sen, "Symbolic execution for software testing: three decades later," *Commun. ACM*, vol. 56, pp. 82–90, Feb. 2013.

[6] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat, "SEEP: exploiting symbolic execution for energy-aware programming," *SIGOPS Oper. Syst. Rev.*, vol. 45, pp. 58–62, Jan. 2012.

[7] S. Schubert, D. Kostic, W. Zwaenepoel, and K. Shin, "Profiling software for energy consumption," in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pp. 515–522, 2012.

[8] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*, EuroSys '11, (New York, NY, USA), pp. 153–168, ACM, 2011.

[9] J. Moreno, J. Wenninger, J. Haase, and C. Grimm, "Energy profiling technique for network-level energy optimization," in *AFRICON, 2011*, pp. 1–6, 2011.

[10] Z. Zhang and S. Fu, "Macropower: A coarse-grain power profiling framework for energy-efficient cloud computing," in *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, pp. 1–8, 2011.

[11] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in tinyos," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, (Berkeley, CA, USA), pp. 1–1, USENIX Association, 2004.

[12] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, (Piscataway, NJ, USA), IEEE Press, 2005.

Operating System Energy Accounting

Andreas Mosthaf
Friedrich-Alexander University Erlangen-Nuremberg
andreas.mosthaf@e-technik.stud.uni-erlangen.de

ABSTRACT

Besides performance, energy is an important issue for operating systems—especially when they are running on a mobile platform. Targeting an optimal lifetime of the system battery, operating systems have to provide mechanisms for energy accounting and energy controlling. This document describes the prerequisites for an effective energy management. It also introduces ECOSystem and the Cinder operating system, two implementations of an energy-aware operating system, along with applications developed for these systems that demonstrate the benefits of energy management on operating systems.

1. INTRODUCTION

Since mobile phones—the dominating mobile systems on the market—are capable of running general-purpose operating systems like Linux, there is a great demand for making operating systems more energy-aware. Improvements have been made to make energy consumption more visible by means of smart device interfaces. In addition, modern devices provide a lot of energy-saving modes to avoid wasting energy. These improvements allow operating systems a coarse control of the platforms' energy behavior.

Energy management requires energy accounting on the one side and energy controlling on the other side. This document describes the prerequisites for an effective energy management on operating systems. An overview over methods and techniques, providing relevant parameters concerning energy resources, is given in Section 2. Section 3 then describes the required power model and control mechanisms. Section 4 and Section 5 contain a brief introduction to ECOSystem [5] and the Cinder operating system [3]—two implementations of energy aware operating systems. Thereafter, Section 6 describes three applications, developed on the Cinder operating system, demonstrating the benefits of an energy aware operating system in typical use cases.

2. ENERGY VISIBILITY

Energy management on operating system level requires at first precise information about energy consumption of the managed devices on the one hand and characteristics of the battery on the other hand. Although modern hardware components may provide interfaces to retrieve the relevant parameters, in most cases measurements are necessary to get accurate results.

Due to the closed nature of mobile systems like smartphones, measuring the power consumption of individual, functional units is difficult. In such cases measuring the total system power and estimating the consumption of single components will lead to inaccurate results. However, there are mechanisms that allow operating systems to work around this kind of error.

Discharge Rate	Usable Battery Capacity (normalized to 1C discharge rate)
C/5	107 %
C/2	104 %
C	100 %
2C	94 %
4C	86 %

Table 1: Characteristics of a lithium-ion battery, adapted from [1].

2.1 Battery Characteristics

Battery lifetime is an important factor for mobile systems and therefore it is a primary target of energy management to maximize it. The discharge rate of a battery has a nonlinear impact on the usable battery capacity. High discharge rates can lower the capacity to 70 %–80 %. Table 1 shows the characteristics of a typical lithium-ion battery.

The Smart Battery interface in the ACPI specification [2] allows operating systems to query all relevant values like current voltage, remaining capacity, and discharge rate. Unfortunately, queries to this interface are slow and return only averaged values of power consumption.

2.2 Power Consumption Measurement

To measure the power consumption of managed devices one has to consider the various states a device could reach. The consumption of a processor, for example, depends on the usage of the built-in functional units and therefore on the current workload. Additionally modern devices offer power saving modes we also have to consider. The transition between the different modes of such devices is also a source of energy consumption that has to be taken into account.

Table 2 shows the measured energy consumption of an IBM Travelstar 12GN harddisk. Spinup, spindown and accessing a block on the disk drains a fixed amount of energy, whereas energy consumption in one of the three idle states depends on the time the harddisk spends in the respective state.

State	Cost	Time Out (Sec)
Access	1.65 mJ/Block	
Idle 1	1600 mW	0.5
Idle 2	650 mW	2
Idle 3	400 mW	27.5
Standby (disk down)	0 mW	
Spinup	6000 mJ	
Spindown	6000 mJ	

Table 2: Measured power state values and time-out values of an IBM Travelstar 12GN hard disk, adapted from [3]

2.3 Energy Profiling

To provide precise energy accounting we have to assign the measured energy consumption to the application that causes it. A simple approach is to sample power consumption and to assign the sample values to the thread currently using the processor. The results will be inaccurate due to the fact that simultaneously active hardware components could have been triggered by two different applications.

A more precise method to account energy online makes use of performance counters embedded in modern processors. Performance counters are implemented as hardware registers. By registering events that imply the consumption of a certain amount of energy at these counters, the operating system is able to change its behavior and therefore to adapt the expected power drain. For example, a high number of main memory references, although only a little number of instructions were executed, could indicate that performance is dominated by the main memory latency. Therefore, throttling the processor speed will improve the energy efficiency without a significant negative impact on system performance.

3. ENERGY CONTROL

Targeting a given battery lifetime means a limitation of the discharge rate of the battery, shown in Table 1. Based on the information provided by hardware measurements, energy-aware operating systems must be able to adapt their scheduling so that the systems' overall power consumption never exceeds the demanded discharge rate. To provide energy-aware scheduling, mechanisms for accounting and distribution of available energy resources among all competing tasks are required.

3.1 Energy Model

A prerequisite for energy accounting is an abstraction for the resource energy. Such an abstraction is provided by the *Currentcy Model* [5]. The model uses a common unit to account and allocate energy. The word currentcy is made up of the words currency and current since this unit is used to pay for a certain amount of current. As the operating system targets a certain consumption rate, one unit represents the right to consume a certain amount of energy in a fixed time interval.

3.2 Control Mechanisms

Energy management on operating systems requires mechanisms for accurate accounting and effective distribution of the resource energy. The following subsections describe three basic mechanisms: isolation, delegation, and subdivision.

3.2.1 *Isolation*

Isolation is an important security mechanism of process and memory management in operating systems. Since applications should not be able to drain energy from other applications, isolation is therefore an important factor for energy-management, too.

3.2.2 *Delegation*

Delegation allows a task to loan its available energy to other tasks. Therefore this is an important mechanism of inter-application cooperation. Donor and recipient of a delegation are both able to consume the delegated resource. By delegating resources from more than one donor to a single recipient, applications are able to contribute to expensive operations.

3.2.3 *Subdivision*

Subdivision allows a task the partitioning of its available energy. Combined with delegation, subdivision enables tasks to loan parts of their energy to other tasks. This is especially important for applications with child processes. In this case the main task might want to be assured that child processes are not able to starve other components of the application.

4. ECOSYSTEM

ECOSystem is an implementation of the currentcy model, based on the Linux operating system. For handling energy as a first class resource, the kernel was modified to provide mechanisms for allocating and accounting energy. This includes a resource container object for the storage of energy units. The prototype of ECOSystem was evaluated on an IBM Thinkpad T20 laptop with a set of microbenchmarks targeting the CPU, the disk, and the network interface.

4.1 Currentcy Allocation

ECOSystem provides an interface to allow the user to set a target battery lifetime. The maximum discharge rate for the battery depends on the targeted lifetime, as shown in Section 1.1. The difference between the current and the maximum discharge rate determines the amount of energy that can be allocated in a fixed time interval. This value corresponds to an amount of energy units that can be distributed between all competing tasks. The distribution of energy is performed by a periodic kernel thread that modifies the values of all tasks' resource containers periodically. If a task does not use all of its energy units, it can accumulate these units up to a certain limit.

4.2 Currentcy Accounting

Performing system operations, like the execution of an instruction or requesting data from a disk, requires a certain amount of energy. If a task wants to perform such an operation, the operating system checks the corresponding energy value. If the value is sufficiently high, the operation will be executed. Otherwise, the task is blocked until it accumulated enough energy units for the requested operation.

Currentcy Model

App	CPU (mJ)	HD (mJ)	Net (mJ)	Total (mJ)
DiskW	430	339,319	0	339,749
NetRecv	256,571	0	553,838	810,409
Compute	8,236,729	0	0	8,236,729

Program Counter Sampling

App	CPU (mJ)	HD (mJ)	Net (mJ)	Total (mJ)
DiskW	430	16	24	470
NetRecv	256,571	9,235	20,206	286,012
Compute	8,236,729	326,404	531,789	9,094,922

Table 3: Energy accounting: currentcy model vs. program counter sampling, adapted from [5].

Due to the numerous, different energy characteristics of managed devices, ECOSystem uses energy charging policies with respect to these differences:

- CPU: Execution of instructions on the processor is charged in dependence of a fixed power value and the processing time. A task running out of energy units will be preempted until it accumulates more units.
- Hard Disk: In addition to a fixed amount of energy for every block accessed, all tasks that accessed the disk within the same session share the costs for spinning up and down the disk.
- Network Interface: Sending or receiving of data packets is charged in dependence of the transmit power, the size of the packet, and the bitrate.

Assigning the costs of CPU operations to the correct task is quite simple, whereas tracking the energy consumption of the disk or the network interface is more complex. Files in ECOSystem are accessed through file related system calls. Hereby the container ID of the calling task is stored within the buffer cache entry of the disk. When the cache entry is actually written, the appropriate resource container will be charged for the operation. Source tasks of network operations are identified by their associated source socket.

4.3 Energy Accounting

The result of an evaluation of the effectiveness of energy accounting implemented in ECOSystem is shown in Table 3. In this experiment three benchmarks were run simultaneously on the system, each addressing a separate functional unit. DiskW writes 4 KB of data every four seconds to the disk, NetRecv receives continuously data from the network interface at the highest bitrate, and Compute is a batch job running continuously on the processor. For comparison Table 3 shows the results of a similar run using a program counter sampling technique.

ECOSystem assigns energy consumption to the appropriate application, whereas the results of the program counter sampling show significant accounting errors. With program counter sampling, energy consumption is measured periodically and the values are assigned to the task currently running on the CPU. The error in Table 3 arises from device operations currently being executed but triggered by a task that it is not running.

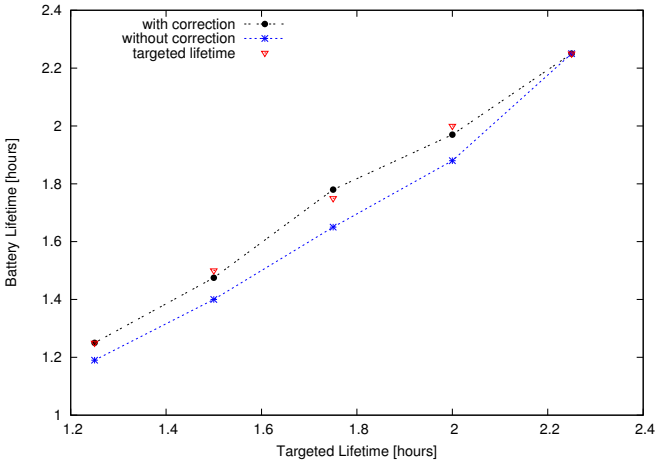

Figure 1: Targeted battery lifetime compared to the measured lifetime, adapted from [5].

4.4 Battery Lifetime

The primary design objective for the development of ECOSystem was to achieve a targeted battery lifetime. As mentioned in Section 2, offline measurements of closed mobile systems could be inaccurate. This experiment shows a possible impact of such an error on the battery lifetime.

In this experiment the system runs a CPU-intensive microbenchmark. Additionally an accounting error is modeled by decreasing the power consumption value for the usage of the CPU. The usage of the CPU is now cheaper than it should be and will lead to a shorter lifetime of the battery. Figure 1 shows the achieved lifetime of the system battery in comparison to the targeted lifetime.

To handle such errors the operating system can make use of the Smart Battery interface. By periodically checking the remaining capacity of the battery, the discharge rate of the battery and hence the amount of allocated energy can be dynamically adapted.

4.5 Energy Sharing

This experiment demonstrates that ECOSystem is able to manage energy sharing between simultaneously running applications. One of the applications is ijpeg, a CPU-intensive graphical application. The other application is netscape, which is mainly using the network interface. A performance indicator for ijpeg is the computation speed and therefore its CPU utilization. The performance of network applications like netscape depends on response times and therefore on page load latencies. Table 4 shows the measurement results for various energy ratios between these two applications, whereby the amount of allocated energy for both applications is set to 5 W.

The measurements show that both tasks match their targeted allocation. Although netscape makes use of all three functional units, ECOSystem tracks the power consumption across all devices accurately. The CPU utilization of ijpeg, and therefore its performance, increases nearly equivalently to the allocated power, whereas netscape shows a non-linear behavior due to non-linear energy characteristics of the network interface.

a)

Energy Share	Power Alloc (W)	Ave Power Used (W)	CPU Util (%)
70%:30%	3.5	3.507	22.55%
60%:40%	3.0	3.008	19.34%
50%:50%	2.5	2.500	16.08%
40%:60%	2.0	2.008	12.91%
30%:70%	1.5	1.503	9.67%
20%:80%	1.0	1.005	6.46%

b)

Energy Share	Power Alloc (W)	Ave Power Used (W)	Page Delay (s)
70%:30%	1.5	1.49	29.205
60%:40%	2.0	2.006	17.441
50%:50%	2.5	2.457	9.928
40%:60%	3.0	2.961	6.322
30%:70%	3.5	3.443	3.934
20%:80%	4.0	3.663	3.032

Table 4: Results of energy sharing between ijpeg (a) and netscape (b), adapted from [5].

5. CINDER OPERATING SYSTEM

Cinder is an operating system designed for modern mobile phones. It extends HiStar [4], a secure operating system that provides *containers* and *gates* in addition to conventional kernel objects.

Every object in HiStar has to be referenced by a container including containers ifself. The hierarchical structure of containers is used by the Cinder operating system to deallocate resources and therefore to implement a mechanism for delegating and subdividing resources. Gates provide secure inter-process communication by using security labels for all objects. This security concept of HiStar ensures isolation of containers.

The Cinder operating system extends HiStar with two additional kernel objects: *reserves* and *taps*. These two extensions to the kernel are explained in the following paragraphs.

5.1 Reserves

Reserves are kernel objects that permit the usage of energy. To determine the amount of energy a task is allowed to use, a power model similar to the currentcy model is used. The value of a reserve is therefore a permission to use a certain amount of energy in a fixed time interval. If the value of a reserve is not high enough for the operation the corresponding task wants to perform, the kernel will prevent execution.

Although it is possible to partition a reserve's value and assign it to other reserves, this is in most cases not practical, because threads rarely need to delegate fixed amounts of energy. Usually threads provide other threads with energy by using a fixed rate.

5.2 Taps

In order to guarantee a certain battery lifetime the discharge rate of the battery has to be limited. Transferring energy between reserves with a fixed rate is therefore a better approach than delegating fixed amounts of energy.

Taps are kernel objects that transfer a certain quantity of energy between two reserves with a fixed rate. Therefore a tap contains a rate, a source reserve, and a sink reserve. Instead of transferring a fixed amount of energy, proportional taps transfer a certain fraction of the source reserves' energy to the sink reserve. In practice, taps are implemented as a thread whose job is to transfer energy values between corresponding reserves periodically.

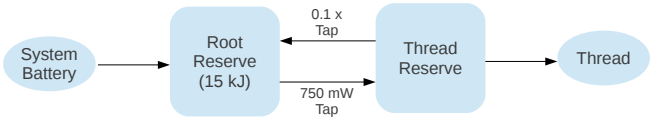

Figure 2: Consumption graph, adapted from [3].

5.3 Energy Consumption Graph

Due to the hierarchical structure of reserves and taps, it is suitable to model it with a directed graph. The root of such a directed graph represents the reserve connected to the system battery; all other reserves are a subdivision of this root reserve. A simple example of such a directed graph is shown in Figure 2. In this example the root reserve is connected to the 15 kJ system battery. A single application draws energy from a separate reserve connected to the root reserve over a 750 mW tap.

5.4 Energy Hoarding

Tasks can accumulate unused amounts of energy to use it for future operations. However, to improve the performance of the whole system, it is good practice to reclaim unused energy and assign it to other tasks. This problem can be solved by using backward taps. A backward tap transfers a fraction of the accumulated energy back to the source reserve. The fraction of the backward tap is also a limitation for the amount of energy the sink reserve is able to hoard. In Figure 2, a backward tap transfers 10 % of the reserve's accumulated energy back to the root reserve. When the sink reserve level in the example reaches 700 mW, both taps have the same rate and therefore the level cannot raise anymore.

6. APPLICATIONS

Improving the battery lifetime of a mobile system is just one goal of energy management. Another target is the performance the user expects. A common use case of modern smartphones is to play music while browsing the internet. In this case user experience depends on the delay of network requests and a disruption-free playback of the music file. Although a user may accept varying delays, a disruption of the playback will have an unacceptable impact on his experience.

The following subsections describe applications that were developed on the Cinder operating system using reserves and taps. They represent typical use cases operating systems have to handle and will show the ability of this operating system to control energy on mobile systems.

The platform on which Cinder is implemented and the following applications are evaluated is an HTC Dream, which is also known as Google G1. Because of its closed, integrated structure measuring becomes a challenging task. Unfortunately the ARM11 core processor provides no performance

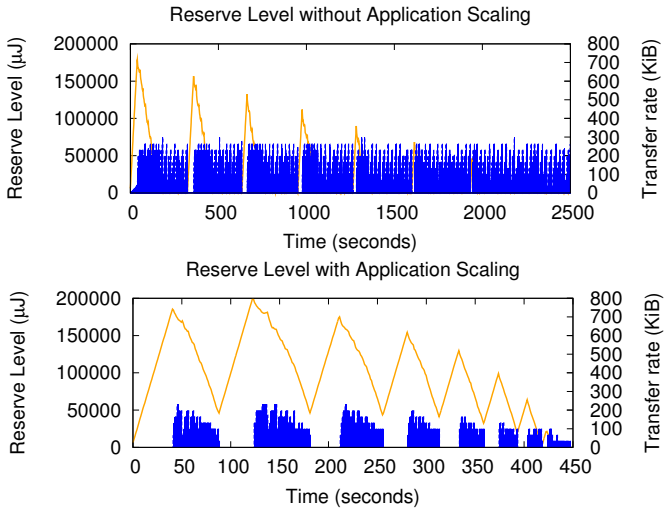

Figure 3: The results of the image viewer application, adapted from [3].

counters. Therefore energy accounting is based on offline measurements of device power states.

6.1 Energy Aware Applications

Reserves and taps allow developers fine-grained control of resources. Adapting the image quality of a graphical application to a given energy level is a good example for an energy aware application.

The application described here is a network picture gallery. A thread separate from the main thread with a separate reserve handles the downloading of pictures. The rate of the tap between the reserves of the main application and the download thread depends on the frequency of image requests and the size of the pictures. By periodically checking the resources' energy level the application is able to adapt its behavior if the downloader is consuming energy too quickly. In this case the application requests only partial data from the interlaced PNG images.

Figure 3 shows the results of test runs with and without energy aware scaling where a batch of image requests is made periodically. The line represents the energy level of the downloader thread's reserve, and the bars show the amount of data downloaded per image. A pause between the batch requests allows the reserve of the downloader to accumulate energy when using scaling. Without scaling the transfer rate stays constant until the reserve runs out. This leads to a disruption of the downloader until the reserve has accumulated enough energy again. Therefore this example makes a trade-off between constant transfer rates and picture quality.

6.2 Background Applications

Background applications are another challenge for energy management. Despite being invisible for the user they may interfere with foreground applications and could therefore have a negative impact on user experience.

The example in this section models an RSS reader and a mail client. Each application has a reserve connected to a

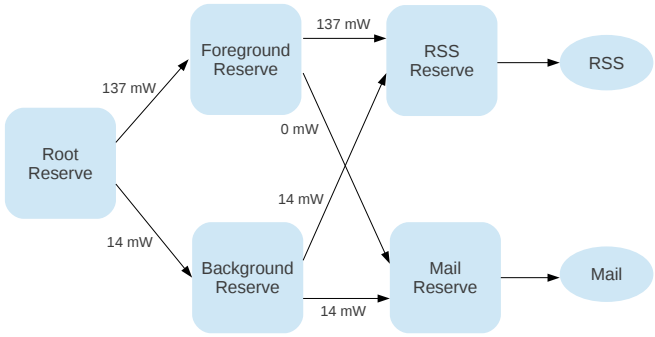

Figure 4: The energy consumption graph for the background application, adapted from [3].

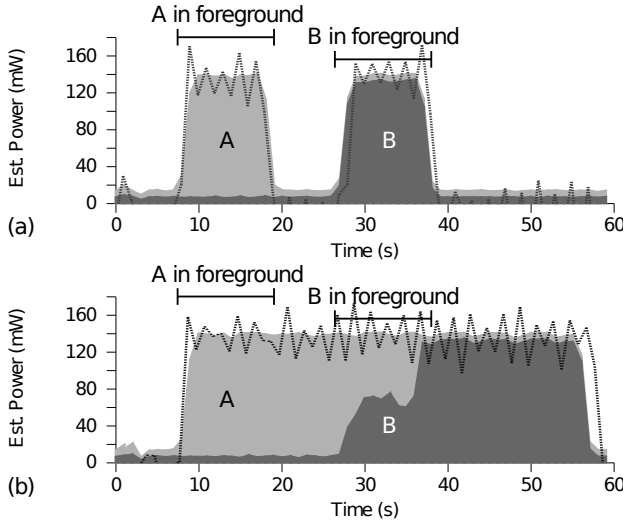

Figure 5: The results of test runs with two tasks spinning from foreground to background, adapted from [3].

foreground and a background reserve, as shown in Figure 4. The taps to the background reserve have a constant small rate of 14 mW for both applications. The rates of the taps to the foreground reserve depend on the current state of a task. The task manager sets the foreground task tap to a rate of 137 mW, which is the power the processor requires to run. The rate of the background task is set to 0 mW and therefore, the background task is not able to run the processor until it accumulates enough energy from the background reserve.

Figure 5a shows the results of a test run. The dark areas in Figure 5 show the power estimation for the two running tasks and the dotted line is the result of an actual power measurement. Task A runs in the foreground in the 10 s till 20 s interval, task B between 30 s and 40 s. Figure 5b shows a test run with a higher foreground rate of 300 mW. With the rate exceeding the required processor power, task A is able to accumulate the unused energy. Therefore task A has still enough energy to use the processor although set to the background and interferes with task B until energy runs out after 40 s.

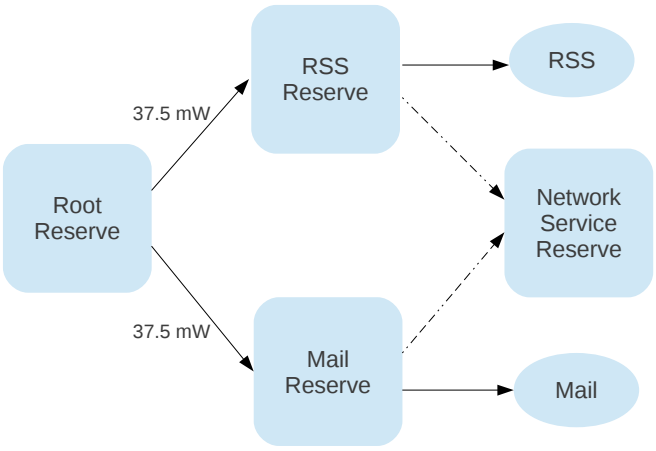

Figure 6: The energy consumption graph for the network application, adapted from [3].

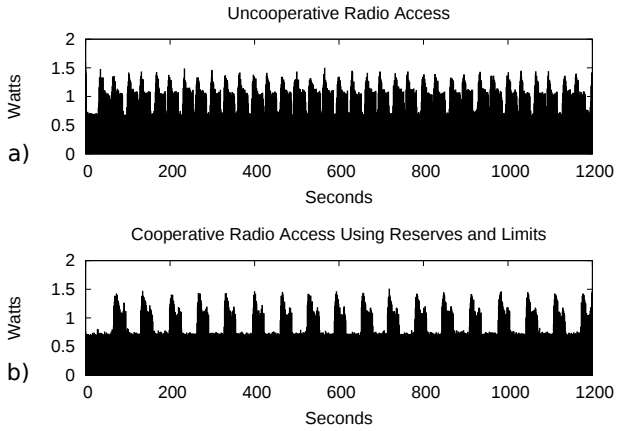

Figure 7: The results of test runs with (a) and without (b) a coordinated network stack, adapted from [3].

6.3 Network Applications

Network interfaces are very expensive regarding their energy consumption. This is aggravated by the fact that they have a non-linear energy characteristic. Initial activation costs of a radio interface are very high but can be compensated through bulk transfers. A common use case are background applications that make periodic use of the network interface.

This example describes the network access of two applications over a network daemon, shown in Figure 6. Both applications should make a poll every 60 seconds. Giving each task enough energy to activate the network daemon every 60 seconds leads to the result shown in Figure 7a. Polling uses the network interface just for a few seconds. The time between the polls of the two different applications is therefore sufficient for the network interface to switch back to its idle mode and both tasks have to pay the full activation costs for their polls.

A better approach is to implement the network daemon with its own reserve to which the polling tasks can delegate their energy to. To activate the network interface, the sum of the reserves' energy of the calling task and the network daemon must reach the activation costs. If the sum is not high enough, the calling task delegates its energy to the daemon's reserve and blocks until enough energy is accumulated. Both of the polling tasks now only get enough energy to activate the network interface if they work in unison. The result of this implementation is shown in Figure 7b. Both tasks are still polling every 60 seconds but now the network interface is activated only once in this period which leads to a significantly decreasing overall energy consumption.

7. CONCLUSION

The performance of a mobile platform depends highly on the lifetime of the system battery. Improving energy management is therefore an important topic of research, especially considering that energy is a more and more limited and expensive resource.

Abstracting energy to a first class resource and providing mechanisms to isolate, delegate, and subdivide it enables operating systems to manage energy on a fine-grained level. The evaluation of the presented applications developed on the Cinder operating system and ECOSystem shows that these systems are able to reach a given battery lifetime. Furthermore, these systems allow developers to implement energy aware applications that will improve the user experience by adapting the applications' behavior to the current energy level.

However, it is uncertain if this approach improves the overall energy efficiency, since there were no measurements made to compare energy consumption, performance and battery lifetime of the evaluated systems with standard systems. Providing the ability to account and control energy on such a fine-grained and accurate level requires a lot of additional computation time that causes higher energy consumption and a lower system performance. Consequently, it is doubtful if managing energy as a first class resource is a promising approach to improve energy efficiency on mobile platforms.

8. REFERENCES

[1] Bellosa, F. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop* (2000), pp. 37–42.

[2] Intel Corporation and Microsoft Corporation and Toshiba Corporation. *Advanced Configuration and Power Interface Specification.* http://www.teleport.com/acpi, 1996.

[3] Roy, A., Rumble, S. M., Stutsman, R., Levis, P., Mazieres, D., and Zeldovich, N. Energy management in mobile devices with the Cinder operating system. In *Proceedings of the European Conference of Computer Systems 2011* (2011), pp. 139–152.

[4] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazieres, D. Making information flow explicit in HiStar. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (2000), pp. 263–278.

[5] Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. ECOSystem: Managing energy as a first class operating system resource. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (2003), pp. 123–132.

Runtime Environments and Software Frameworks

Jeremias Isnardy
Friedrich-Alexander University Erlangen-Nuremberg
JeremiasIsnardy@web.de

ABSTRACT

The increasing need for mobile computer systems has engendered a new priority that has been neglected so far: awareness of the system's energy consumption. Not only smartphones, tablets or laptops should run as long as possible, but also perpetual systems using environmental energy, which should be designed with energy-awareness in mind. Common programming languages do not provide possibilities for the programmer to adapt her code with records to energy. This document presents runtime environments and software frameworks designed to support the programmer at writing energy-aware code: *Eon* [11], a programming language and runtime system for perpetual systems and *EnerJ* [10], a Java extension introducing approximate data types.

1. INTRODUCTION

Energy consumption of computer devices has become a main concern for developers with many challenges. The question, what a computer system can accomplish is closely connected to the availability of energy resources. Several approaches have been made to reduce energy consumption on low-power architectures, performance/power trade-offs and resource management, which can be applied all without knowing the software. However, the programmer does not have the opportunity to influence these approaches. Another way to manage the energy more efficient, is to allow energy considerations at the level of programming languages.

This can be derived in different ways. The following Section 2 demonstrates one possible approach how to provide functionality of managing energy for perpetual systems. These systems are a special kind of computer systems. They are designed in a way that they could run indefinitely just by harvesting energy of the environment; like solar or wind energy. Eon provides a programming language and runtime system for such systems, so that the programmer can implement different scenarios depending on the available amount of energy. The section introduces the principle of Eon, explains its functionality as a programming language and as a runtime system, shows experiments and evaluation results.

Section 3 shows a different more general approach. EnerJ is an extension to the programming language Java and introduces a new approximate data type, which consumes less energy then common data types. The section explains the idea of EnerJ, how it is implemented and what hardware it needs.

2. EON: A LANGUAGE AND RUNTIME SYSTEM FOR PERPETUAL SYSTEMS

Eon is both, a programming language and a runtime system, designed especially for the development of perpetual systems. According to the authors Eon was the first energy-aware programming language (2007). It is based on the domain-specific programming language Flux [2], which allows programmers to build programs from different languages, including C and nesC. nesC stands for network embedded systems C, an extension to the C programming language, used to build applications for platforms using TinyOS.

2.1 Concepts of Eon

Perpetual systems have the property of running indefinitely. To do that, such systems are harvesting environment energy, e.g., motion, sun, wind or heat differentials. Nevertheless, harvesting energy sources and developing a perpetual system face several challenges. Either energy is not always available or it can be fluctuating. Depending on the purpose of the system, energy costs are also influenced by the environment dynamically.

The main idea of Eon is to let the developer choose which parts of the program are important and therefore should run continuously, and which parts can be deactivated when energy is low. In that way the system can react to environmental changes automatically. Eon provides a simple way to associate particular control flows with abstract energy states, representing the current available energy combined with an approximation for the near future. The Eon runtime system executes only those flows that are chosen for the given energy state, managed by an integrated *automatic energy management*. The energy states are described as a relative size ("high", "low"), thus the programs are portable to other hardware platforms with different energy profiles.

2.2 Eon Programming Language

Eon is a *coordination language* [5] that combines code of conventional languages, like Java, C or nesC. This way it is easy to separate code concerning energy from program logic. Reusing existing code and porting Eon programs to other platforms are additional advantages. A coordination language describes the flow of data through different components.

Eon programming language uses the following structures:

- **Source Node:** Basically the leading program instance providing important data.

- **Concrete Node:** Corresponds to functions implemented in C or nesC. Required set of input data is provided by the source nodes. It is producing a set of output arguments.
- **Abstract Node:** Describes the flow of control and data through multiple concrete or abstract nodes.
- **Conditional Flows** through **Predicated Types**, which need to be implemented by the programmer. The types are choosing one of the possible execution paths.
- **Eon Power States:** Eon is able to perform runtime adaptations. For this it is necessary so specify an adaptation policy as a collection of adjustments organized in a state ordering. Every state contains a list of adjustments, sorted by their priorities.
- **Adaptive Timers:** To save energy the most common adjustment is to disable components periodically. It is necessary to define a range of intervals from which the timer can choose.
- **Energy-State Based Paths:** Similar to the conditional flow an execution path will be chosen, but in this case depending on the energy state.

Figure 1 shows a condensed example of Eon source code. The code was written for a GPS tracking device. The Eon programming language does not affect any program logic, but it manages the flow of the program, depending on choices of the programmer or of the current energy state. The code was designed to update GPS data in an appropriate rate and log them.
It uses source nodes, like `ListenBeacon` (line 7) or `GPSTimer` (line 8) to get the important required data and to provide them for the concrete nodes, like `GetGPS` (line 12). The actions of these nodes are implemented in C or nesC and are not of any concern for Eon. The abstract node `GPSFlow` (line 37) navigates the flow of the data, in this case from `GetGPS` to another abstract node `StoreGPSData` (lines 38-39). This one is affected by the predicated type `gotfix`. Depending on it, `StoreGPSData` calls either `LogGPSData` (line 14) or `LogGPSTimeout` (line 16). Line 46 and 47 are defining a path, based on the energy state, which has been defined in line 32 for high power. Depending on this energy state `GPSTimer` is getting updated in an interval between 1 hour and 10 hours, for the high energy state. Otherwise only every 10 hours if the energy is low.

```
1  // Predicate Types
2  // SYNTAX: typedef PRED_TYPE PRED_TEST
3  typedef gotfix TestGotFix;
4
5  // Source Node Declaration
6  // SYNTAX: NODENAME () => (OUTPUTS);
7  ListenBeacon() => (msg_t msg);
8  GPSTimer() => ();
9
10 // Concrete Node Declaration
11 // SYNTAX: NODEAME (INPUTS) => (OUTPUTS);
12 GetGPS() =>
13         (GpsData_t data , bool valid);}
14 LogGPSData(GpsData_t data bool valid)
15         => ();
16 LogGPSTimeout(GpsData_t data bool valid)
17         => ();
18 LogConnectionEvent(msg_t msg) => ();
19
20 // Regular Sources
21 // SYNTAX: source NODENAME => NODENAME;
22 source ListenBeacon => HandleBeacon;
23
24 // Timer Sources
25 // SYNTAX: source timer NODENAME
26                 => NODENAME;
27 // Eon Timer Source
28 source timer GPSTimer => GPSFlow;
29
30 // Eon States
31 // there is always an implicit BASE state
32 stateorder {HiPower};
33
34 // Abstract Nodes and Predicate Flows
35 // SYNTAX: ABSTRACT[[type ,..][state]] =
36 // CONCRETE ->...CONCRETE;
37 GPSFlow = GetGPS -> StoreGPSData;
38 StoreGPSData:[*,gotfix][*] = LogGPSData;
39 StoreGPSData:[*,*][*] = LogGPSTimeout;
40
41 // Abstract Node using Energy Predicates
42 HandleBeacon:[*,*][HiPower]
43                 = LogConnectionEvent;
44
45 // Eon Adjustable Timer
46 GPSTimer:[HiPower] = (1 hr, 10 hr);
47 GPSTimer:[*] = 10 hr;
```

Figure 1: Eon Code [11]

2.3 Eon Runtime System

It is not sufficient to have a programming language supporting energy-aware coding only. It is also necessary to have a runtime system assembling it correctly. Choosing the ideal energy state for the system can be very sensitive.

The algorithm Eon uses to determine the ideal energy state tries to achieve the highest possible fidelity with the given energy while avoiding two energy states: empty (even high priority flows can not be executed) and full (harvested energy can not be saved). The runtime system attempts to find the ideal power state that is suitable as long as possible. Initially, it is assumed that the system runs at the highest energy state and computes the amount of consumed and produced energy over a short interval T_i. If the computation results in an empty battery, the system lowers the energy state and repeats the computation. Once T_i has been found, it examines longer intervals in the form of $2^n \cdot T_i$ for $n = 1..N$ to make sure it is truly sustainable.

To adapt the energy state in the correct way, an accurate model of the energy consumption is required, containing energy costs and frequency of each execution path or flow. Each time a flow completes, an exponentially weighted moving average (EWMA) of the flow's energy cost gets updated.

Additionally, an estimate of how much energy the system is going to harvest in the future is required. Assuming that the energy production of the following days will be similar to the recent days the prediction algorithm from Kanasl et al.[6, 7] provides sufficiently accurate predictions.

2.4 Experiment

To evaluate Eon, various deployments were conducted, including an automobile tracking system. Five GPS receivers, gaining their energy from the sun, were mounted on top of the roof of cars and have been tracked for two weeks. During this time the weather was highly variable, thus also the energy supply.

The collected data were used afterwards in a trace-driven simulation extending the period of two weeks to three months to get an impression of the long-term behavior of Eon. To avoid transient deviations, only the results of the last month of the simulation were considered. Several test cases were simulated, each with a different GPS sampling rate according to five energy policies:

- **Conservative** – minimum sustainable rate of all traces
- **Greedy** – maximum sustainable rate of all traces
- **Best Static** – best sustainable rate for each trace
- **Eon** – using a solar predictor algorithm
- **Eon (Oracle)** – using a perfect weather predictor

2.5 Results

Figure 2 shows the average number of daily GPS readings, comparing the five different traces and the five different policies. Every trace was exposed to a different amount of sun energy, thus there are big variations among the traces.

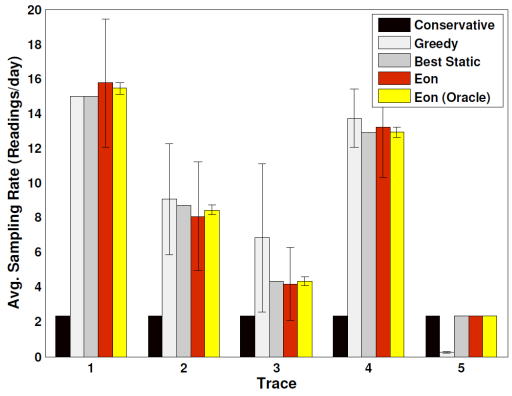

Figure 2: Average number of daily GPS readings for different energy policies and traces [11]

It should be mentioned that the best sustainable policy and the oracular Eon policy only exist in theory. Both would require an exact prediction of solar trends. Regarding this fact, the Eon predictor achieves a surprisingly good result compared to the oracle. However, it is important to note that the ratio between energy consumption and battery size is quite low. Thus, errors of the prediction algorithm do not have much impact on this deployment.

Figure 3 shows the result from another perspective. It compares the amount of each trace's energy that has been consumed by different parts of the system. The percentage number represents the average amount of time the trace was spending with an empty battery. The board overhead is the energy spent in the measurement board, the idle energy is the energy spent waiting and not executing a flow, the GPS energy was spent on taking samples, unused energy was energy left in the battery and wasted energy is any energy that was collected but could not be stored due to a fully charged battery.

Figure 3 also demonstrates the advantages of Eon towards the conservative and greedy policies. The conservative one

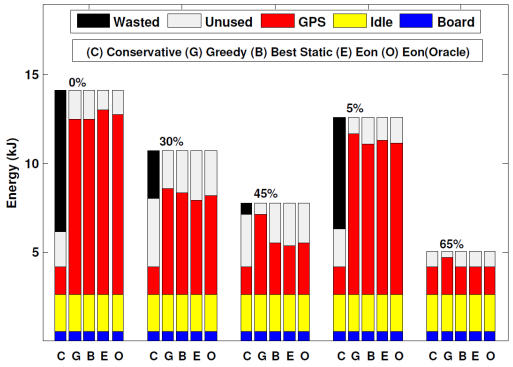

Figure 3: Amount of energy consumed by different parts of the system [11]

has a low sampling rate of GPS data and wastes a lot of collected energy. The greedy policy on the other hand has large periods of dead time. Eon accomplishes both, a good sampling rate and a good workload of the energy without having any dead time.Additional experiments were conducted with Eon, generating several further results.

A user study examined Eon regarding its usability. In it the effort of coding a program with Eon and the performance of this code were compared with the same task written in C. The participants of the Eon group did not have any difficulties to learn the ropes of Eon and finished the task in nearly the same time as the C coders. Regarding the performance of the program the big advantage of Eon emerged. For the Eon coders it was quite easy to organize their program in a more efficient and in a more energy-aware way, respectively.

Eon needs additional computation to determine the ideal energy state. This computations need also some energy. For this reason Eon needed to be examined regarding a possible overhead. Some experiments showed that this needed additional energy is so small compared to the remaining energy consumption that it can be neglected.

As mentioned before the capacity of the battery has also an impact. If the capacity is small, errors of the prediction algorithm can lead to dead times in the worst case. Figure 4 demonstrates that the smaller the battery size the bigger the risk of a dead time. From a battery size of approximately 150 mAhr this risk can be neglected.

3. ENERJ: APPROXIMATE DATA TYPES

EnerJ is an extension to Java adding approximate data types.

3.1 Concept of EnerJ

Many applications or some sections in an application can tolerate inaccuracies. For these regions precise computations are a waste of energy. EnerJ tries to exploit this fact and introduces a new data type, especially designed to perform computations not in a precise but an approximated way. It introduces type qualifiers that distinguish between approximate and precise data types and claims to be safe and general.

3.2 Type System

This section lists the extensions of EnerJ to Java and how to deal with it as a programmer. It outlines the changes made

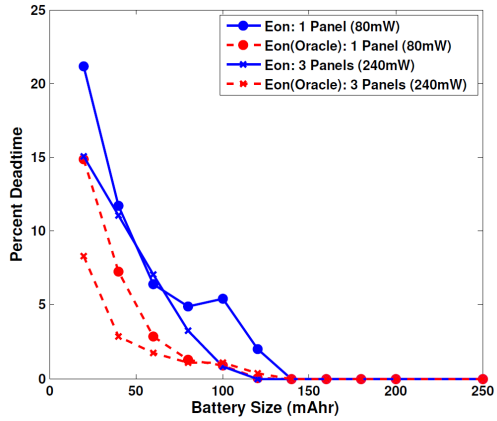

Figure 4: Dead time of a device with different battery sizes [11]

to Java and indicates several possible pitfalls.

Type Annotations

Every value has an approximate or a precise type, which the programmer can assign by using `@Approx` and `@Precise` (since it is default, not necessary) respectively. It is not desirable and thus not possible to assign an approximate-typed value into a precised-typed one. Only the opposite direction is valid.

In some cases it can be necessary to work temporarily with precise even when using approximated data. The programmer has the option to use the *endorsement* concept, introduced by [1]. This allows her to use approximated data as precise ones, as the following code example shows [10].

```
@Approx int a = ...;
int p;  // precise by default
p = a   // illegal
p = endorse(a)  // legal
```

Operations

When mixing the computations of precise and approximated data, additional features need to be considered. This is achieved by overloading operators and methods based on the type qualifiers. To ensure the correct choice of the operator, EnerJ implements a bidirectional type checking [3]. So the computation of the approximated value a = b + c (both precise) returns an approximated value, even if the parameters b and d are both precise.

Control Flow

Approximated data cannot be used in conditions, since it could affect the control flow and thus violate the property that no information flows from approximated to precise data [10]:

```
@Approx int val = ...;
boolean flag;   // precise
if(val == 5) { flag = true; }
else { flag = false; } //illegal
```

By using the introduced endorse concept, the programmer is able to work around this restriction.

Objects

It is also possible to create approximable classes. Clients have the option to create precise and approximable instances of it. To guarantee this possibility the `@Context` qualifier has been introduced. All variables marked with it can be precise or approximated, depending on the choice of the programmer, as illustrated in the following code example [10]:

```
@Approximable class IntPair {
     @Context int x;
     @Context int y;
     @Approx int numAdditions = 0;
     void addToBoth(@Context int amount) {
          x += amount;
          y += amount;
          numAdditions++;
     }
}
```

This class has parameters x and y which can be both precise and approximated and another parameter `numAdditions` which is always an approximated one. The parameter amount of the function `addToBoth` is also depending on the programmer's choice.

It is not only possible to distinguish parameters but also whole methods. The programmer needs to write one for the precise case and one for the approximated one, for example [10]:

```
@Approximable class FloatSet {
  @Context float[] nums = ...;
  float mean() {
     float total = 0.0f;
     for (int i = 0; i < nums.length; ++i)
        total += nums[i];
     return total / nums.length;
  }

  @Approx float mean_APPROX() {
     @Approx float total = 0.0f;
     for (int i = 0; i < nums.length; i += 2)
        total += nums[i];
     return 2 * total / nums.length;
  }
}
```

To distinguish methods overloaded on precision the suffix `_APPROX` has been introduced. Both methods are calculating the mean value of all numbers saved in `nums`. In case of the approximated method, the calculation averages only half of the numbers, saves computation steps and therefore also energy.

3.3 Hardware Realization

It is not sufficient to distinguish between approximated and precise types at the programming level. Except for some computation steps one can save in an algorithm, there will not be any energy savings if the hardware does not support some kind of separation on the hardware level. Thus an approximation-aware execution substrate is needed.With the right choice of the hardware both, approximated storage and approximated operations can be used.

Approximated storage uses unreliable registers, data caches and main memory. The register number respectively the memory address separates it from the precise data. The physical memory reserves regions for the approximated data. The approximated operations use specific instructions, which can use special functional units to perform them.

Figure 5 illustrates a hardware model providing approximation-aware functionality. The shaded areas represent com-

ponents that support approximation. Registers and data cache use SRAM storage cells, which can be made approximate by decreasing supply voltage. Functional units support it in the same way, whereas floating point functional units additionally can decrease the size of their mantissas. Main memory (DRAM) supports it by reducing the refresh rate.

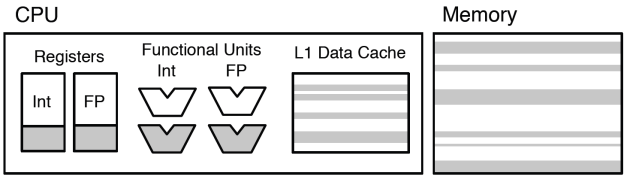

Figure 5: Approximation-aware hardware model [10]

Table 1 summarizes three different policies for the hardware settings to examine the amount of energy being saved by using approximated data. Numbers marked with * are claimed to be educated guesses by the authors, the others were taken from literature. For every policy different adaptations are used with an increasing impact to the hardware settings. E.g. the mild policy uses only small changes, obtaining more accuracy but losing potential to save energy.

	Mild	Medium	Aggressive
DRAM refresh: per-second bit flip probability	10^{-9}	10^{-5}	10^{-3}
Memory power saved	17%	22%	24%
SRAM read upset probability	$10^{-16.7}$	$10^{-7.4}$	10^{-3}
SRAM write failure probability	$10^{-5.59}$	$10^{-4.94}$	10^{-3}
Supply power saved	70%	80%	90%*
`float` mantissa bits	16	8	4
`double` mantissa bits	32	16	8
Energy saved per operation	32%	78%	85%*
Arithmetic timing error probability	10^{-6}	10^{-4}	10^{-2}
Energy saved per operation	12%*	22%	30%

Table 1: Three different approximation strategies [10]

3.4 Experiment

In order to evaluate EnerJ, a compiler and runtime system has been implemented to execute benchmark as if it was running on an approximation-aware architecture. The chosen benchmarks are listed in Table 2. They are existing Java programs, modified with the approximated data type.

To gain any information about the energy savings, some assumptions have to be made. No overheads are considered when implementing or switching to approximate hardware, like a latency when scaling the voltage. In addition only a simplified model with three components is influencing the energy consumption:

- **Instruction execution:** To estimate the savings, abstract energy units are assigned to arithmetic operations. These estimations are based on three studies [4, 8, 9]
- **SRAM storage:** SRAM storage and instructions that access it are assumed to account for 35% of the microarchitecture's power consumption. The rest of it is consumed by the remaining instruction executions. To compute the total CPU power savings both savings are scaled accordingly.
- **DRAM storage:** A server-like setting has to be assumed, in which DRAM accounts for 45% of the power and CPU 55%.

Application	Lines of code	Prop. FP	Total decls.	Annot. decls.	Endorsements
FFT	168	38.2%	85	33%	2
SOR	36	55.2%	28	25%	0
MonteCarlo	59	22.9%	15	20%	1
SMM	38	39.7%	29	14%	0
LU	283	31.4%	150	23%	3
ZXing	26171	1.7%	11506	4%	247
jMonkeyEng.	5962	44.2%	2104	19%	63
ImageJ	156	0.0%	118	34%	18
Raytracer	174	68.4%	92	33%	10

Table 2: Applications used as benchmarks [10]

Considering the fact that various generous assumptions are necessary to gain any information about the energy savings the results should be considered as very optimistic.

3.5 Results

The functions listed in Table 2 are annotated manually with the attempt to strike a balance between reliability and energy savings. The first five applications are scientific algorithms from the SciMark2 benchmark. ZXing is a bar code decoder for smartphones, jMonkeyEngine a game engine, ImageJ a raster image manipulation software and Raytracer a 3D image renderer. The validity of these benchmarks is hard to evaluate. It can not be assumed that the authors used the best possible adaptations for the applications. By all means it would had been a better approach to use less benchmarks and to vary the amount of approximated data in it.

Figure 6 shows the normalized behavior of the energy consumption of all these applications. The bar labeled "B" represents the baseline value: the energy consumption without approximated data types. The numbered bars correspond to the different strategies introduced in Table 1. Every bar is divided into several regions, each representing the individual portion of the energy consumption.

Each benchmark can be computed with less amount of energy, depending on the aggressiveness of the used strategy. But saving energy is not the only aspect which need to be considered. Computing with approximated values instead of precise ones leads to unavoidable errors in the results. Figure 7 compares these errors of each benchmark regarding the used strategy. Some of the results show huge errors, even for the medium strategy, whereas other ones only show neglectable deviances. No matter what, the use of approximated data types will always be a tradeoff between quality of data and energy savings.

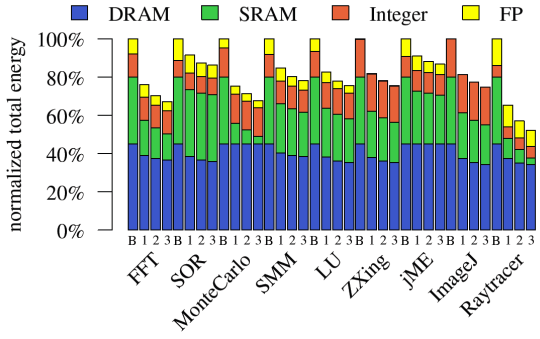

Figure 6: Estimated CPU/memory system energy consumed for each benchmark [10]

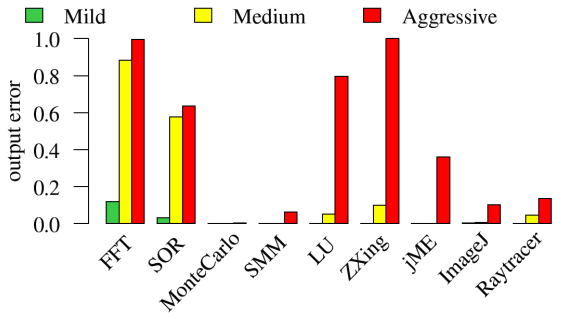

Figure 7: Output errors of the benchmarks depending on the strategies [10]

4. CONCLUSION

Developing computer systems that are not only fast and reliable but also energy-aware is one of the big challenges in computer science today. Creating runtime systems and software frameworks that allow the programmer to affect the energy consumption directly is inevitable with respect to the future. This document presents two different ways to get more control of the energy flow in the system.

Eon, designed especially for perpetual systems, uses a quite simple approach. It allows the programmer to determine different scenarios for different energy states, so it is up to him to decide how the system behaves having much energy or less. To ensure the correct implementation, Eon is not just a programming language but also a runtime system. So it is able to manage the desired code but always with respect to the available energy. It is mainly designed for perpetual systems and for these it is a suitable way to manage available energy. But it should be mentioned that it saves energy only at the expense of performance. It does not consider any ways to save energy in the system directly e.g. through hardware adaptations.

EnerJ tries to save energy of a system from a different point of view. It introduces a new data type as an extension to Java. This kind of data is only approximated so it does not need the same size of storage and is faster in computation. In this way the usage of this new types can save a lot of energy. As easy as the approach sounds, it is quite difficult to integrate it in computer systems. The hardware of the system has to be designed for it and the determination of the region in an application where it could be useful can by quite tricky.

Both approaches use different main ideas. But both are associated with the mentioned disadvantages. Energy savings may have become a main aspect for developers, but there are a set of other aspects a programmer need to consider. Does the additional work pay off? Is the concept one that is sustainable or will it vanish in the near future? Is the program still portable with the used concept or does it need a specified hardware? How much effort would it be to integrate the concept into an existing project? A programmer need to ask herself all these questions before using on of these introduced approaches or another one. There is and will always be more to it than just saving energy.s

5. REFERENCES

[1] Askarov, A., and Myers, A. A semantic framework for declassification and endorsement. In *ESOP* (2010).

[2] Burns, B., Grimaldi, K., Kostadinov, A., Berger, E. D., and Corner, M. D. Flux: A language for programming high-performance servers. In *Proceedings of USENIX Annual Technical Conference* (May 2006).

[3] Chlipala, A., Petersen, L., and Harper, R. Strict bidirectional type checking. In *TLDI* (2005).

[4] D.Brooks, V.Tiwari, and Martonosi, M. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA* (2000).

[5] Gelernter, D., and Carriero, N. Coordination languages and their significance. In *Communications of the ACM* (Februar 1992).

[6] Kansal, A., Hsu, J., Srivastava, M. B., and Raghunathan, V. Harvesting aware power managment for sensor networks. In *Design Automation Conference* (July 2006).

[7] Kansal, A., Jason Hsu, S. Z., and Srivastava, M. B. Power management in energy harvesting sensor networks. In *ACM Transactions on Embedded Computing Systems* (May 2006).

[8] Li, S., Ahn, J. H., Strong, R., Brockman, J., Tullsen, D., and Jouppi, N. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO* (2009).

[9] Natarajan, K., Hanson, H., Keckler, S. W., Moore, C. R., and Burger, D. Microprocessor pipeline energy analysiss. In *ISLPED* (2003).

[10] Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., and Grossman, D. EnerJ: Approximate Data Types for Safe and General Low-Power Computation. In *Proceedings of Programming Language Design and Implementation (PLDI '11), California* (June 2011).

[11] Sorber, J., Kostadinov, A., Garber, M., Brennan, M., Corner, M. D., and Berger, E. D. Eon: A language and runtime system for perpetual systems. In *Proceedings of The Fifth International ACM Conference on Embedded Networked Sensor Systems (SenSys '07), Sydney* (Nov. 2007).

Entwurf umweltfreundlicher Rechenzentren

Wolfgang Rödle
Friedrich-Alexander-Universität Erlangen-Nürnberg
wolfgang.roedle@mb.stud.uni-erlangen.de

KURZBESCHREIBUNG

In der vorliegenden Ausarbeitung zum Thema umweltfreundliche Rechenzentren geht es um Ansätze zur Reduzierung des Energieverbrauchs großer verteilter Systeme und Rechnerverbunde. Dabei wird genauer auf das Gebäudedesign am Beispiel des in Lockport (New York) erbauten Yahoo! Compute Coop-Rechenzentrums eingegangen. Die Bauform, die Innenarchitektur, die Ausrichtung und die klimatischen Bedingungen des Standorts sorgen für einen geringeren Energieverbrauch, indem dadurch auf aktive Kühlung (z. B. Lüfter oder Wasserkühlsysteme) verzichtet werden kann. Die Kühlung der Computer erfolgt hierbei fast ausschließlich durch einströmende Außenluft. Das Gebäudedesign von Yahoo! reduziert auch den CO_2-Ausstoß und den Wasserverbrauch bzw. verringert die Abwassermenge, welche bei den mit Wasser gekühlten Anlagen anfällt. Ein weiterer Vorzug des Designs sind die kürzeren Bauzeiten und damit geringeren Baukosten des Rechenzentrums, die durch die Einfachheit und das verbaute Material ermöglicht werden.

Energieersparnis durch Gebäudearchitektur reicht jedoch nicht aus, um ein Rechenzentrum umweltfreundlich zu gestalten. Das GREEN-NET-Framework bietet hier eine Möglichkeit an, zusätzlich Energie durch verbesserte energiegewahre Software-Infrastruktur zu sparen. Der Ansatz versucht Endbenutzern ein Gefühl für den Energieverbrauch ihrer Programme zu verschaffen und zieht diese aktiv in Entscheidungen über Leistung und Energieverbrauch mit ein.

1. EINFÜHRUNG

Da Internetdienste immer mehr Funktionen bereitstellen und immer mehr Benutzer diese in Anspruch nehmen, sei es von einem Desktop-Computer von zu Hause aus oder mit einem Smartphone von unterwegs, werden auch mehr Rechenzentren benötigt, die viel Strom verbrauchen. Der hohe Energieverbrauch kommt u. a. von ineffizienten Kühlsystemen und von betriebsbereiter IT-Ausrüstung, die über längere Zeit ungenutzt bleibt. Die vorliegende Ausarbeitung fasst Möglichkeiten der Stromeinsparung von Rechenzentren zusammen und geht dabei speziell auf die Gebäudearchitektur vom Yahoo!-Compute-Coop-Rechenzentrum sowie auf GREEN-NET ein. Letzteres Framework ermöglicht einerseits, dass Rechenzentren energieeffizienter betrieben werden können, und andererseits deren CO_2-Ausstoß zu verringern, der durch mit Kohleenergie angetriebene Lüfter und Kühlsysteme anfällt. Bei der Gebäudearchitektur wird hauptsächlich der Strom durch Weglassung dieser Lüfter und Kühlsysteme reduziert, wobei auf der Software-Seite bei den Internetdiensten der Strom durch Abschalten von unbenutzten Applikationen und Netzknoten sowie durch direkten Zugriff auf die Hardware, wie CPUs oder Festplatten, realisiert wird.

Im zweiten Kapitel wird das „grüne“ Rechenzentrum von Yahoo! vorgestellt. Dabei wird genauer auf die Thermodynamik und die Kühlmethoden der Anlage eingegangen und anschließend Vergleiche zu ähnlichen Rechenzentren von Google und Facebook genannt. Das dritte Kapitel befasst sich mit einem Ansatz für Internetnetze und Clouds, die durch Einbeziehung der Endbenutzer energieeffizienter laufen sollen. Das vierte Kapitel liefert eine Diskussion über die vorgestellten Konzepte. Abschließend fasst das fünfte Kapitel kurz die Möglichkeiten den Stromverbrauch bei Rechenzentren zu reduzieren zusammen.

2. YAHOO! COMPUTE COOP (YCC)

Das YCC-Rechenzentrum wurde aufgrund des Klimas in Lockport errichtet. Dabei handelt es sich um ein Rechenzentrum, dessen Server zu 99 % durch Außenluft gekühlt werden, d. h. es fallen keine bzw. sehr geringe Kosten für die Klimatisierung an. Durch die spezielle Bauweise und die Ausrichtung der einzelnen Gebäude werden thermodynamische Gesetze und die an diesem Standort günstigen Windbedingungen hervorragend ausgenutzt und sorgen dadurch für eine umweltfreundliche und kostenfreie Kühlung („Free Cooling“) der Computer. Neben der Reduzierung vom Stromverbrauch wird auch der Wasserverbrauch im Vergleich zu konventionellen, wassergekühlten Rechenzentren drastisch verringert. Der Wasserverbrauch sinkt gegenüber konventioneller Rechenzentren auf 5 %. Auch der CO_2-Ausstoß, der durch Dieselmotoren angetriebene Lüfter entsteht, wird reduziert, da sich die Anzahl der Lüfter auf ein Minimum beschränkt. Der Standort Lockport ist nicht nur aufgrund seines Klimas geeignet, sondern auch aufgrund der Nähe zu den Niagarafällen, die sich 30 Kilometer westlich der Anlage befinden. Damit kann die IT-Ausrüstung durch nachhaltige Energie betrieben werden. Diese Eigenschaften machen das YCC zu einem umweltfreundlichen, „grünen“ Rechenzentrum. Zusätzlich gewähren die Gebäudearchitektur, das Baumaterial und die Bauweise sowohl kürzere Bauzeiten und somit geringere Baukosten als auch eine kostengünstigere Instandhaltung der Rechenzentren.

Die Designidee des YCC bringt viele Normenüberschreitungen der American Society of Heating, Refrigerating and Air-Conditioning Engineering (ASHRAE) mit sich und auch Risiken. Experten sind der Meinung, dass die Vorzüge die Risiken bei weitem übersteigen und das Design in Zukunft Standard wird. Yahoo! besitzt bereits viele Patente für das YCC und deren Technologien.

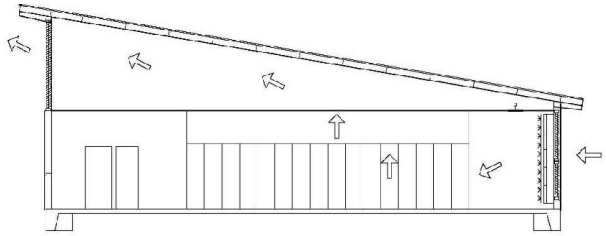

Abbildung 1: Luftstrom bei unklimatisierter Außenluftkühlung [2]

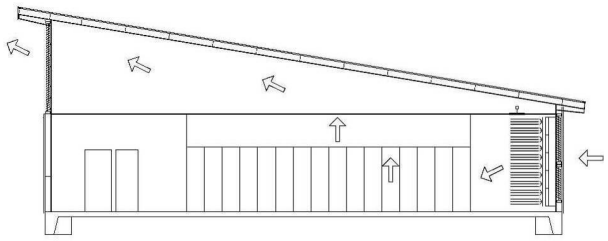

Abbildung 2: Luftstrom bei klimatisierter Außenluftkühlung [2]

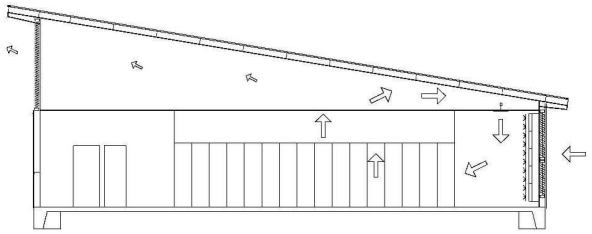

Abbildung 3: Luftstrom bei Außen- und Innenluftkühlung [2]

2.1 Kühlmethoden

Das Gebäude nutzt eine Eigenschaft der Thermodynamik aus, welche besagt, dass wärmere Luft aufsteigt und Luftmassen unterschiedlicher Temperatur sich auszugleichen versuchen. Um eine geeignete Architektur zu finden, baute Yahoo! anfangs ein Testlabor in Santa Clara (Kalifornien) [1]. Nachdem dieses sehr gute Ergebnisse lieferte, wurde das Compute-Coop-Projekt im Jahr 2009 gestartet. Die Abbildungen 1, 2 und 3 zeigen die Bauform einer Hälfte des Gebäudes. Im oberen Bereich, dem Dachboden, der einzelnen Gebäude ist ein leerer Raum, der die durch die Server erwärmte Luft aufsteigen lässt. Das Schrägdach sorgt für den Abzug zur Mitte hin, wo sich ein Kamin befindet. An den oberen Wänden des Kamins und der unteren Außenwand des Gebäudes befinden sich Lüftungsschlitze mit Filtersystemen, um die für die Kühlung erforderliche Außenluft einströmen zu lassen und diese von Schmutz- und Staubpartikeln zu befreien. Sensoren liefern Daten über die Windgeschwindigkeit und die Lufttemperatur, die dann an Überwachungsgeräten ausgewertet werden. Anschließend wird mit Hilfe dieser gewonnenen Daten einer der drei möglichen Kühlmodi der Anlage ausgewählt. Zusätzlich wird die Geschwindigkeit der einströmenden Luft durch Dämpfungslüfter geregelt. Damit die Computer die Luft zur Klimatisierung nutzen können, müssen sie in offenen Racks (dt. Regalen) gelagert werden [1, 4]. Das Rechenzentrum wurde großflächig angelegt, sodass eine optimale Raumluftzirkulation gegeben ist. Frühere Forschungen ergaben, dass Temperaturen zwischen 16 und 18 °C optimal für Datenzentren sind. Erfahrungswerte zeigten jedoch, dass Temperaturen von 27 °C vollkommen ausreichend sind [10]. Die drei bereits angesprochenen Kühlmodi werden im Folgenden vorgestellt.

2.1.1 Vollständig unklimatisierte Außenluftkühlung

Bei Temperaturen von 21 °C bis 29,5 °C werden in diesem Modus die Rechner nur durch Außenluft gekühlt. Diese strömt durch Lüftungsschlitze an der äußeren Wand ein und wird dort gesäubert. Durch den leeren Raum im oberen Bereich des Gebäudes steigt die durch die Server erwärmte Luft und wird durch die oben liegenden Lüftungsschlitze wieder nach außen geleitet. Hierbei entsteht kein Overhead, also keine zusätzlichen Stromkosten durch Lüftersysteme.

2.1.2 Vollständig klimatisierte Außenluftkühlung

Bei Temperaturen von 29,5 °C bis 43,5 °C wird die Luft kurz nach dem Einströmen in das Gebäude durch ein wassergesättigtes Medium geleitet, um Verdunstungskühlung zu ermöglichen. Durch die Wasserpartikel in der Luft werden den Servern zusätzlich Wärme durch Verdunstung entzogen (siehe auch [5]).

Nur knapp neun Tage im Jahr befinden sich die Temperaturen in diesem Bereich und die Anlage benötigt zum Klimatisieren der Rechner diese zusätzliche Verdunstungskühlung. An den restlichen Tagen im Jahr reicht die Außenluft zur Kühlung der Rechner aus.

2.1.3 Gemischte Außenluftkühlung

Im Winter bzw. bei Temperaturen unter 21 °C wird ein Teil der in den oberen, leeren Raum geleiteten Luft wieder nach unten gepumpt und mit der einströmenden Außenluft gemischt, um hohe Temperatur- und Luftfeuchtigkeitsunterschiede zu verhindern. Dafür werden Ventilatoren an der Decke nahe der Außenwände eingesetzt.

2.2 Vergleich mit anderen Rechenzentren

Tabelle 1 zeigt einen Vergleich von Yahoo!s Rechenzentren. Die Einrichtung in Santa Clara wird mit Wasser gekühlt und braucht damit zusätzlich Platz für die Wasserspeicherung und -leitungen sowie Energie für den Transport des Wassers. Die Anlage in Quincy (Washington) wird auch durch Außenluft gekühlt, hat jedoch nicht die vorteilhafte Bauform des YCC, um die Außenluft ohne viel Bearbeitung, das heißt ohne Ventilatorsysteme, zur Kühlung der Server zu verwenden. Es ist zu erkennen, dass die Anlagen in Lockport und Quincy durch neue Technologien deutliche Energieeinsparungen für die Server, also die IT-Ausrüstung, besitzen. Der Energieverbrauch für die Kühlsysteme wurde bereits bei der Anlage in Quincy durch Ventilatoren statt Wasserkühlung stark verringert. Das YCC senkt diese nochmals um 90 %. Die Effizienz ist – auch wenn es sich bei diesen Daten nur um erste Zahlen handelt – sichtbar.

Um ein übersichtliches Maß für den Energieverbrauch und die Effizienz der Rechenzentren zu haben, wird das Maß PUE

	Santa Clara, CA	Quincy, WA	Lockport, NY
Leistungsaufnahme pro Server (W)	257	101	99
Leistungsaufnahme für 25.000 Server ohne Kühlung (kW)	6.438	2.529	2.489
Leistungsaufnahme für die Klimatisierung (kW)	2.285	434	46
Gesamte Leistungsaufnahme für 25.000 Server (kW)	**8.722**	**2.963**	**2.535**

Tabelle 1: Analyse der Energieeffizienz von verschiedenen Rechenzentren von Yahoo! [1]

(Power usage effectiveness, dt. Effektivität der Stromnutzung) eingeführt [1, 11]:

$$\text{PUE} = \frac{\text{Gesamter Energieverbrauch der Anlage}}{\text{Energieverbrauch für die IT}}$$

Dieses stellt den gesamten Energieverbrauch der Anlage ins Verhältnis zu dem Energieverbrauch, der ausschließlich durch die IT anfällt. Zum Energieverbrauch der IT zählen u. a. Energieverbräuche von Rechnern, Speichereinheiten und Netzwerken. Der gesamte Energieverbrauch setzt sich aus den Verbräuchen der IT und aus den Energieverbräuchen der Kühlsysteme, Kraftstoffnutzung, Stromverluste und sonstigen Energieverbräuchen zusammen. Aus der Tabelle 2 geht deutlich hervor, dass dieses Verhältnis bei der YCC-Anlage beinahe bei dem Wert 1,0 liegt und somit fast keine zusätzlichen Energiekosten außerhalb der IT anfallen.

Die großen Einsparungen beim Kohlenstoffdioxidausstoß und bei der Energie sind der Tabelle 2 ebenfalls zu entnehmen. Sie zeigt die umweltfreundliche Seite des Gebäudedesigns.

Der PUE-Wert ist aufgrund seiner Neuheit nicht exakt definiert. Die meisten Organisationen wie Yahoo! oder Facebook nehmen in den Gesamtverbrauch nur den gesamten Verbrauch der Anlage auf, berücksichtigen jedoch nicht Verluste im Stromnetz, bei Transformatoren oder bei Generatoren. Wenige Unternehmen nehmen diese Verluste ebenfalls im PUE mit auf. Im Vergleich dazu hat das neue Rechenzentrum von Google einen PUE-Wert von 1,12, jedoch unter Einbezug sämtlicher Verluste. Mit der allgemeineren Rechnung liegt der PUE-Wert mit 1,06 sogar unter dem des Yahoo! Compute Coop [9]. Die vorliegenden Werte der neuen Anlage von Facebook schlagen die Rechenzentren von Yahoo! und Google. Dort wird ein PUE-Wert von bis zu 1,02 angegeben, was somit fast dem Idealwert entspricht. Aktuelle Werte sind mit regelmäßigen Updates unter [12] abzulesen. Sie schwanken um den Wert 1,09.

Das Datenzentrum von Facebook in Prineville funktioniert genau wie das YCC durch Kühlung der Server mit Außenluft und Wasserspritzdüsen für die heißen Tage im Jahr. Für die „open racks“ (dt. offene Computerregale) hat Facebook das „Open Compute Initiative“-Projekt gestartet [3] und sich zu der besonderen Bauform der Racks die zugehörigen Patente gesichert. Zudem kann die Anlage in Prineville Regenwasser aufbereiten, nutzt Solarenergie und leitet die von den Servern ausgestrahlte Wärme in Büroräume, um diese zu heizen [8].

	Santa Clara, CA	Quincy, WA	Lockport, NY
PUE	1,62	1,27	1,08
Relative Energieeinsparung für ein 9-MW-YCC-Gebäude (kWh/Jahr)	26.541.307	12.094.773	-
Durchschnittliche jährliche Kohlenstoffdioxideinsparung (Tonnen CO_2)	14.863	6.773	-

Tabelle 2: Energie- und Kohlenstoffeinsparung der Rechenzentrumdesigns im Vergleich [1]

Egal ob Yahoo!, Google, Facebook oder andere Unternehmen, alle Unternehmen, die groß angelegte Rechnersysteme für Internetdienste nutzen, achten auf eine energieeffiziente und umweltfreundliche Gestaltung dieser Anlagen. Dabei werden natürliche Ressourcen, wie Flüsse, Wasserfälle oder Sonnenlicht, herangezogen, um saubere, nachhaltige Energie zu nutzen.

3. GREEN-NET-FRAMEWORK

Energie- und Kohlenstoffdioxideinsparungen durch umweltfreundliche Kühlsysteme reichen im Allgemeinen nicht aus, um den Energieverbrauch bei Rechenzentren zu minimieren und diese als umweltfreundlich und energieeffizient einzustufen. Ein Framework mit dem Namen GREEN-NET bietet die Möglichkeit Energie zu sparen durch das Einbinden des Enbenutzers und dessen Entscheidungen. Das GREEN-NET-Framework zeichnet sich durch einen Top-down-Ansatz, der aus drei Stufen besteht, aus. Zunächst soll der Endbenutzer ein Gefühl für den Energieverbrauch der von ihm benutzten Internetleistungen bekommen. Dabei werden Informationen zu den Energieverbräuchen seiner Programme gesammelt und graphisch dargestellt. In der zweiten Stufe soll der Endbenutzer in Entscheidungen zur Reduzierung von Energieverbrauch mit einbezogen werden, wodurch Applikationen und Rechenknoten, die unbenutzt aktiv laufen, abgeschaltet werden. Zudem soll die Hardwareleistung reduziert werden, was hauptsächlich Taktfrequenzen von CPUs und Umdrehungszahlen von Festplatten betrifft. Dabei muss stets ein Kompromiss zwischen Energiereduzierung („grüner Ansatz“) und der Leistung („Performance-Ansatz“) gefunden werden. Da der Endbenutzer diese Entscheidung am ehesten Treffen kann und sollte, wird er im GREEN-NET-Framework mit einbezogen. Die dritte Stufe umschließt die Unterstützung für das automatische Abschalten unbenutzter Netzwerkknoten. Für das Sammeln der Informationen, das Präsentieren der ausgewerteten Daten und die Unterstützung automatisierter Energiereduzierung, werden eigene Tools und Bibliotheken zur Verfügung gestellt. Da es sich hierbei um ein Forschungsgebiet handelt, muss die reale Welt, welche die Endbenutzer mit ihren Entscheidungen umfasst, simuliert werden. Hierfür werden mögliche Entscheidungsrichtungen implementiert, die die Benutzer allgemein treffen können. In den Richtlinien des GREEN-NET-Frameworks sind sechs Abstufungen zwischen Leistung („user“) und maximaler Energieersparnis unter Ein-

haltung gegebener Fristen („deadlined“) vorgesehen.

3.1 Informieren des Endbenutzers

Um den Endbenutzer in Energiesparentscheidungen mit einzubeziehen, muss der Benutzer in erster Linie über den Energieverbrauch seiner Applikationen informiert werden. Dafür bietet zum Beispiel Google das sogenannte PowerMeter an oder Microsoft die Hohm-Cloud-Anwendung [13]. Zusätzlich zum Stromverbrauch kann sogar die Kohlenstoffdioxidemission aufgrund des Energiegemisches des Wohnorts errechnet werden. Es verschafft dem Benutzer ein Gefühl darüber, wie umweltschonend oder -verschmutzend dessen elektrische Geräte betrieben werden. Beide Strommessgeräte beziehen sich jedoch auf ein komplettes Gerät, wie zum Beispiel eine Heizung, Beleuchtung oder einen Computer und nicht auf einzelne Programme. Es müssen auch Daten über den Energieverbrauch der Endbenutzer über Monate und sogar Jahre hinweg gesammelt werden, um den Verlauf und die Entwicklung darzustellen. Das datenschutzrechtliche Problem wird an dieser Stelle offengelegt. Erst nach dem Sammeln und Analysieren können die Strommesser dem Benutzer Vorschläge zum Energieverbrauch liefern.

Die für das hier beschriebene GREEN-NET-Framework benutzte Messmethode misst den Verbrauch eines ganzen Computers in Testlaboren, die ein Rechenzentrum simulieren. Das HAMEG HM8115-2 ist ein Leistungsmesser, der verschiedene Informationen, wie Schein-, Wirk- und Blindleistung und Phase messen kann [14]. Neben dem HAMEG-Leistungsmesser wird noch die Omegawatt box von Omega-Watt für Messungen von Knoten herangezogen. Das Messgerät wird für 162 Knoten verwendet, was dazu dient, den Energieverbrauch verschiedener Knoten zu messen.

Um die gesammelten Daten über den Energieverbrauch einzelner Computer und Internetknoten zu vereinen, wurde eine große Bibliothek implementiert.

Nach Auswertung der gesammelten Daten müssen die Ergebnisse den Endbenutzern zur Verfügung gestellt werden. Dafür gibt es die Möglichkeit die Ergebnisse auf eine Webseite oder mit Hilfe von Internetdiensten dem Benutzer zur Verfügung zu stellen. Die Webseiten werden mit Hilfe des RRDtool-Programms verwendet, um den zeitlichen Verlauf des Energieverbrauchs der Knoten, auf die die Applikationen des Benutzers zugreifen, graphisch darstellen zu können. Dabei werden keine weiteren Informationen über die Nutzung der Programme mitgeliefert. Die zweite Möglichkeit den Energieverbrauch über einen Internetdienst abzufragen, besteht darin einen genauen Zeitraum zu nennen, in dem die Programme auf einen bestimmten Knoten zugreifen.

3.2 Einbezug des Endbenutzers

Bei dem vorgestellten GREEN-NET-Framework besitzen die Endbenutzer zwei Möglichkeiten in den Energieverbrauch aktiv einzugreifen.

- Zum einen können sie explizit angeben, welche Teile der Hardware abgeschaltet werden können, da sie für die derzeitig laufenden Programme weniger relevant sind. Dies bezieht sich rein auf ihre Applikationen. Hierfür muss der Benutzer bei jedem Aufruf eines neuen Programms erneut entscheiden, welche Teile der Hardware abgeschaltet werden können bzw. wie viel Leistung von den jeweiligen Geräten vonnöten ist. Dafür muss der Benutzer genaue Kenntnisse über sein Programm besitzen.
- Die zweite Möglichkeit besteht darin, dass die Endbenutzer der Middleware ein Verhalten vorgeben. Damit nehmen sie Einfluss auf das ganze System und u. a. auch auf Knoten, die komplett abgeschaltet werden können. Das Verhalten schwankt zwischen Energiesparen und hoher Leistung. Bei dieser Möglichkeit muss der Benutzer nicht genau die Hardwarevoraussetzungen seiner Programme kennen.

Je nachdem, wie sich die Benutzer entscheiden, muss die Hardware den Wünschen der Benutzer nach Leistung oder Energieeffizienz eingestellt werden. Dafür gibt es ein Ressourcen- und Auftragsverwaltungssystem (Ressource and Job Management System (RJMS)), das für dieses Framework dynamisch angepasst werden musste – das sogenannte OAR.

Das OAR kann je nach Einstellung bzw. Wunsch des Endbenutzers die Drehzahl der Festplatte und die Taktfrequenz der Prozessoren steuern. Damit können die Benutzer direkt Einfluss auf die Hardware und damit auf den Energieverbrauch der Computer nehmen. Da es sich bei OAR um eine quelloffene Software handelt, sind weitere Entwicklungen, wie das Abschalten von Netzwerkknoten, noch in der Entwicklung.

An dieser Stelle werden bei den Laborversuchen die Entscheidungen der Benutzer über den Kompromiss zwischen hoher Leistung und Energieersparnis durch die oben genannte GREEN-NET-Strategie simuliert. Je nach Modus werden die CPU-Taktfrequenz und/oder die Festplattendrehzahl verringert. Die Skalierbarkeit für weitere Elemente befindet sich auch hier noch in der Erforschung.

3.3 Mechanismus zur automatisierten Sicherstellung von Energiegewahrheit

Der Mechanismus zur automatisierten Sicherstellung von Energiegewahrheit umschließt das automatische Abschalten von Netzknoten oder sogar Computern, die gerade nicht gebraucht werden. An dieser Stelle wird wieder das OAR herangezogen, das diese Funktion automatisch ermöglicht.

3.3.1 Energiereduzierung und Leistungsverluste

Um Leistung und Energieverbrauch gegeneinander abzuwägen, werden zwei Verwaltungsmodi für die anfallenden Aufträge bereitgestellt. Zum einen wird ein Standard-Verwaltungsmodus verwendet, d. h. volle Leistung auch im unbenutzten Zustand, und zum anderen ein stromsparender „grüner“ Verwaltungsmodus, welcher unbenutzte Knoten selbstständig abschaltet und bei Allokationen wieder einschaltet. Beide Verwaltungsmodi wurden bei 50,32 % Systemauslastung (Abbildung 5) und bei 89,62 % Systemauslastung (Abbildung 6) gemessen und verglichen.

Die Abbildungen beschreiben den zeitlichen Verlauf des momentanen Leistungsverbrauchs. Die rote Linie entspricht dem normalen Verwaltungssystem, das durch aktiven Betrieb aller Knoten einen konstanten Verlauf hat. Am Graphen für das grüne Verwaltungssystem, der schwarz eingezeichnet ist, kann man die Ein- und Ausschaltvorgänge der Rechenknoten erkennen. Die Fläche unterhalb der Graphen beschreibt den Energieverbrauch.

In den beiden Abbildungen ist die Energieersparnis des grünen Verwaltungsmodus deutlich zu erkennen. Die hohen

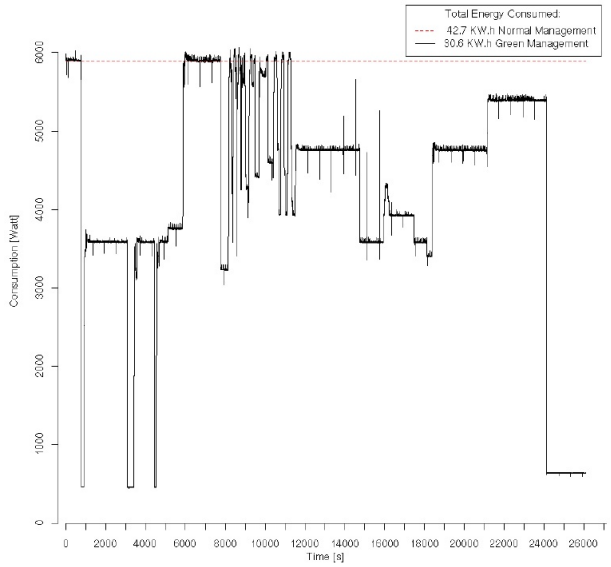

Abbildung 4: Energieverbrauch über die Zeit bei 50,32 % Systemauslastung

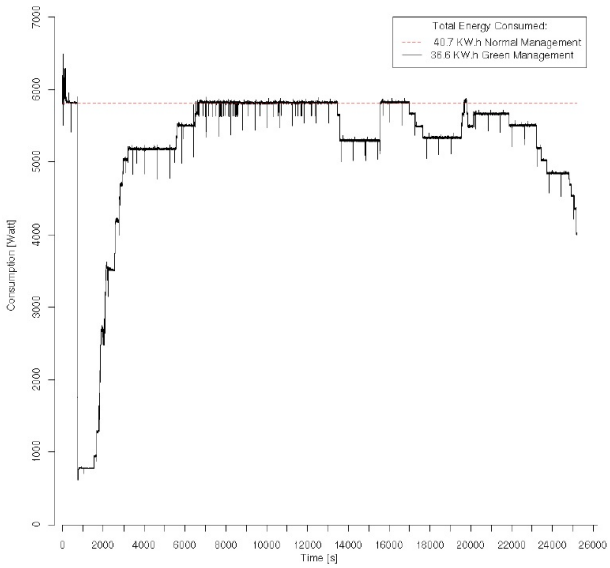

Abbildung 5: Energieverbrauch über die Zeit bei 89,62 % Systemauslastung

Werte am Anfang des Graphen in Abbildung 4 kommen vom Hochfahren aller Knoten vor der Messung. Tabelle 3 stellt die Ergebnisse der Messungen übersichtlich dar. In beiden Fällen ist durch den grünen Verwaltungsmodus ein Energiegewinn abzulesen. Ein Viertel des normalen Energieverbrauchs kann bei 50,32 % der Systemauslastung gespart werden. Im Falle der höheren Systemauslastung ist dieser geringer, da es weniger Zeitfenster gibt, in denen Knoten unbenutzt und damit abschaltbar sind. Jedoch entstehen durch das ständige Ein- und Ausschalten der Knoten extrem hohe Wartezeiten für die Aufträge. Die durchschnittliche Wartezeit mit grünem Verwaltungsmodus bei halber Systemauslastung beträgt knapp 14 Minuten. An dieser Stelle bleibt zu konstatieren, dass das Leistungsvermögen stark unter der Energieersparnis leidet.

3.3.2 Vorhersagen

Um das Problem des Overheads (dt. Mehraufwand) bezogen auf die Wartezeit beim Ein- und Ausschalten der Netzknoten zu verbessern, bieten sich Vorhersagen an, mit deren Hilfe vorausschauend Knoten aktiviert und deaktiviert werden sollen. Dadurch kann einerseits Energie gespart werden und andererseits die Leistung nur marginal beeinflusst werden; im besten Fall bleibt diese gänzlich unbeeinflusst. Hierfür werden u. a. auch Schlupfzeiten (engl. slack times) berechnet und ausgenutzt.

Die Schlupfzeit ist die Zeit zwischen dem voraussichtlichen Terminationszeitpunkt und der Deadline eines Arbeitsauftrags (siehe Abbildung 5) [15]. Sie beschreibt den Spielraum der Bearbeitung und damit die Dringlichkeit des Auftrags. Bei Unterbrechungen z. B. durch Interrupts wird die Fertigstellung des Auftrags verzögert, wodurch sich der Schlupf verringert.

Durch Vorhersagen werden deutlich bessere Ergebnisse im Wettstreit zwischen Energiereduzierung und Leistungsausbeute erzielt.

4. DISKUSSION

Die guten PUE-Werte des YCC sind aus der Tabelle 2 abzulesen, jedoch ist anzumerken, dass das mit Wasser gekühlte Rechenzentrum von Google gleichwertige Zahlen liefert. Es ist also nicht erforderlich das Design von YCC für ein Rechenzentrum heranzuziehen. Auch mit konventionellen, wassergekühlten Anlagen kann das Rechenzentrum durch niedrige PUE-Werte und Wiederverwendung des Wassers umweltfreundlich sein.

Die Kühlmethode des YCC funktioniert nur mit offenen Racks, da sonst das Kühlmittel, also die Luft, nicht die Server erreichen kann. Auch wenn diese Racks immer mehr zu einem Standard werden, sind geschlossene Racks z. B. zum Mieten von Servern nötig, die in solchen Rechenanlagen ohne zusätzlichen Aufwand für die Kühlung nicht eingesetzt werden können.

Wie in [10] beschrieben, liegt die optimale Außenlufttemperatur für die Kühlung der Server bei 27 °C. Lockport bietet sich somit nicht als Standort an, da die Temperaturen nur im Juli durchschnittlich bei den optimalen Werten liegen [16]. Im restlichen Jahr ist es deutlich kühler, sodass die Anlage im dritten Kühlmodus betrieben werden muss, d. h. mit Ventilatoren für die Mischung der kalten Außenluft mit der bereits erwärmten Innenluft. Ein wärmeres, maritimes Klima, wie z. B. in San Diego, ist als Standort für ein Rechenzentrum nach dem Design des YCC vorzuziehen.

Beim GREEN-NET-Framework setzen die Entwickler auf das Verständnis und Wissen der Endbenutzer über ihre Soft- und Hardwarekomponenten. Das ist jedoch für die Benutzung der Geräte nicht vonnöten und somit bei den wenigsten Nutzern vorhanden. Die RJMS-Tools müssen die Benutzer zudem über Spezifikationen der Komponenten informieren bzw. ihnen Vorschläge zur Energiereduzierung liefern. Dies kostet zusätzlich Energie, was weitere Evaluationen erforderlich macht.

Des Weiteren können die Endbenutzer zwischen Leistung und Energieersparnis wählen. Bei dieser Entscheidung werden die meisten Endbenutzer die deutlich kürzeren Wartezeiten

Modus	Normal	Grün	Normal	Grün
Systemauslastung	50,32		89,62	
Anzahl der Aufträge	309		188	
Gesamter Energieverbrauch (kWh)	42,7	30,6	40,7	36,6
Durchschnittlicher Energieverbrauch (kWh)	5,9	4,2	5,8	5,2
Wartezeit der Aufträge im Durchschnitt (s)	8	829	1	218

Tabelle 3: Gesamter Energieverbrauch und Wartezeiten vom normalen und grünen Verwaltungsmodus [1]

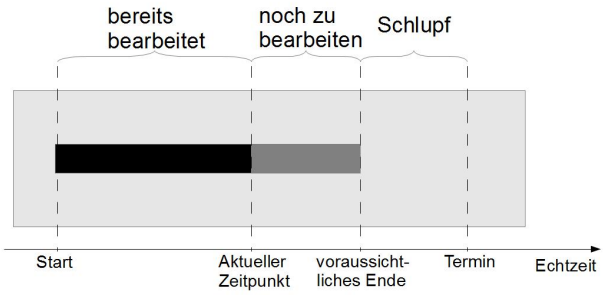

Abbildung 6: Schlupf

(siehe Tabelle 3) und somit die Leistung der Energieersparnis vorziehen. Die hohen Leistungsverluste für größere Stromkostenersparnisse des Endbenutzers und Rechenzentrums sind die Hauptkritikpunkte bei dem vorgestellten Ansatz. Vorhersagen über Ressourcenallokationen sind deshalb unerlässlich, wenn sich dieser Ansatz bei den Endbenutzern durchsetzen soll.

5. ZUSAMMENFASSUNG

Diese Ausarbeitung zeigt Ansätze, die dazu dienen Rechenzentren umweltfreundlicher und energieeffizienter zu gestalten. Das Gebäudedesign von Yahoo! für Rechenzentren hat sich profiliert. Ein PUE-Wert von bis 1,08, 40 % weniger Energieverbrauch für die gesamte Anlage und ein 95 % geringerer Wasserverbrauch im Vergleich zu konventionellen Anlagen sind deutliche Indikatoren, die für eine Umstellung auf die neue Architektur von Rechenzentren sprechen [6, 7]. Nicht nur bei Yahoo!, sondern auch bei anderen Unternehmen wie Google und Facebook, ist diese Architektur für eine nahezu kostenlose Klimatisierung der Rechner heute aktiv im Einsatz und höchster Stand der Technik von Kühlsystemen in Rechenzentren.

Das vorgestellte GREEN-NET-Framework bietet eine Möglichkeit, Energie durch Einwirken auf die Hardware zu sparen. Dabei werden Netzwerkknoten abgeschaltet, Drehzahlen von Festplatten und Taktfrequenzen von Prozessoren individuell eingestellt. Dieser grüne Ansatz zur Energiereduzierung geht jedoch stark auf Kosten der Leistung. Hier sind weitere Entwicklungen nötig und genauere Messdaten über den Stromverbrauch von einzelnen Programmen erforderlich. Auch die individuelle Einstellbarkeit der Hardware benötigt weitere Forschung und Entwicklung. Grundsätzlich ist zu sagen, dass allein das Informieren des Benutzers ein energiebewussteres Verhalten schafft und womöglich den Verbrauch indirekt in Rechenzentren und zu Hause senkt.

6. LITERATUR

[1] Da Costa, G., Dias De Assuncao, M., Gelas, J.P, Georgious, Y., Lefevre, L., Orgerie, A.C, Pierson, J.M., Richard, O, Sayah, A. Multi-facet approach to reduce energy consumption in clouds and grids: The GREEN-NET framework In: ACM/IEEE International Conference on Energy-Efficient Computing and Networking (e-Energy), pp. 95-104. ACM, 2010.

[2] Bob Lytle, Chris Page. Yahoo! Compute Coop (YCC): A next-generation passive cooling design for data centers, Juli 2011.

[3] Mark Hachman. Facebook launches 'open compute initiative' servers, Juli 2011. *http://www.pcmag.com/article2/0,2817,2383257,00.asp*,

[4] Open Racks. Triple racks for servers, 2011. *http://www.opencompute.org/wp/wp-content/uploads/2011/07/DataCenter-Mechanical-Specifications.pdf*

[5] Jay Park. Data center design director, Juli 2011. *http://regmedia.co.uk/2011/04/07/facebook_open_compute_triple_rack.jpg*

[6] Richard L. Brandt. Four companies that went green, Januar 2013. *http://www.greencomputingreport.com/gcr/2013-01-04/four_companies_that_went_green.html?page=2*

[7] Rich Miller. Inside the Yahoo Computing Coop, September 2010. *http://www.datacenterknowledge.com/archives/2010/09-/20/inside-the-yahoo-computing-coop/*

[8] Facebook. Press release: Facebook opens first data center in Prineville, Oregon, April 2011. *https://www.facebook.com/notes/prineville-data-center/press-release-facebook-opens-first-data-center-in-prineville-oregon/10150150581753133*

[9] Google. Effizienz: So machen wir das, 2013. *http://www.google.com/about/datacenters/efficiency-/internal/*

[10] Jürgen Kleinöder, Vorlesung zu energieeffizienten Datenzentren, Erlangen, 2013.

[11] Margarete Rouse. Power Usage Effectiveness (PUE), April 2009. *http://searchdatacenter.techtarget.com/definition/power-usage-effectiveness-PUE*

[12] Facebook. Prineville, OR data center, 2013. *https://www.fbpuewue.com/prineville*

[13] Werner Pluta. Microsoft Hohm hilft beim Stromsparen, Juni 2009. *http://www.golem.de/0906/67997.html*

[14] Hameg. Leistungsmessgerät, Juli 2013. *http://www.hameg.com/1.147.0.html*

[15] Fabian Scheler, Wolfgang Schröder-Preikschat, Real-time systems, 2012,. *https://www4.cs.fau.de/Lehre/WS12/V_EZS/Vorlesung/*

[16] u.s. climate data. Climate of Lockport, NY, 2013. *http://www.usclimatedata.com/climate.php?location=USNY0827*