# The Case for Practical Multi-Resource and Multi-Level Scheduling Based on Energy/Utility

Hermann Härtig, Marcus Völp, Marcus Hähnel
Institute of Systems Architecture, Operating Systems Group
Technische Universität Dresden
Dresden, Germany
{haertig, voelp, mhaehnel}@os.inf.tu-dresden.de

***Abstract*—Energy has become the dominating concern for resource management. We advocate an energy-centered design approach for resource-management systems. To this end, we structure systems in layers, where layers implement higher-level resources using lower-level ones. For each layer, we describe the relation of the performance delivered for the higher layer to its demands on the lower layer and refer to that relation as demand/performance function. The lowest layers are rooted in hardware and express demand in terms of energy, the highest layers provide performance in terms of user-specific utility, thus leading to an Energy/Utility characterization of a complete system.**
**We describe the overall approach, some research challenges and few initial results on the representation of demand/performance functions.**

***Keywords*—energy; utility; scheduling**

## I. Introduction and Overview

Energy usage has become a major if not the dominating concern for the design and implementation of resource management. This leads to additional and different requirements for resource management algorithms. Most current resource management approaches assume resources to be instantly available but scarce. Therefore, especially in the field of embedded and real-time systems, these algorithms attempt to optimize the usage of such instantly available but scarce resources. In contrast, the current or expected technological situation is characterized by abundant availability of inexpensive and heterogeneous latent resources that can be activated using energy which in turn becomes the resource to be considered. This calls for resource management with a step of indirection: energy as the prime resource is used to activate a latent resource or to increase the load on an active resource. If there are alternatives, the choice should be dictated by the energy required.

A simple example for that change in resource management are CPU admission and scheduling algorithms to provide CPU cycles to a task. Most currently used or published algorithms strive to optimize CPU usage for a given set of active CPUs. However, with the appearance of heterogeneous multicore computers and the observation that cores tend to have an energy-optimal usage frequency, there are novel challenges: If a given bandwidth of CPU cycles is required for a task, the available options are 1) to add the task to the load of an active CPU if possible within the optimal frequency or 2) to spend energy to activate a latent CPU or 3) to increase voltage and frequency of an active CPU.

A similar example is the usage of network interfaces. In some application areas, computer systems have several network interfaces with different characteristics, for example one high-speed interface being relatively energy-efficient for high bandwidth and several low-speed interfaces being relatively energy-efficient for low bandwidth. Then, if a certain combined bandwidth is required by the clients of a network stack, there is a choice between using one or few of the low-speed interfaces or the one high-speed interface.

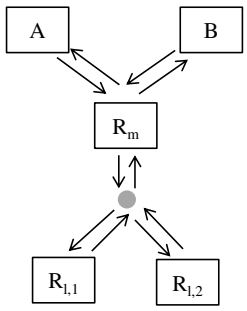


Fig. 1. Choice between alternative low-level resources $R_{l,1}$ and $R_{l,2}$ to generate the performance for the intermediate resource $R_m$

These simple examples have in common that a higher-level resource (CPU or network bandwidth) is provided using alternative lower-level ones (cores or NICs, Figure 1). In both examples, the choice should depend on the energy needed.

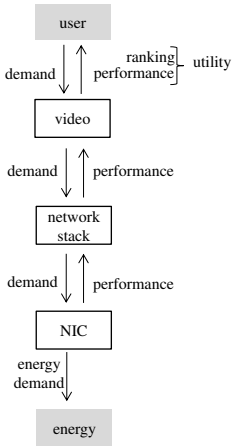


Fig. 2. Energy/utility mapping through demand/performance.

To enable such choices, we advocate to structure systems in layers, where layers implement higher-level resources using lower-level ones (Figure 2). For each layer the performance delivered to the higher layer relative to its demands on the lower layer is known as its demand/performance function. The lowest layers are rooted in hardware and express demand in terms of energy, the highest layers provide performance in terms of user-specific utility, thus leading to an Energy/Utility characterization of a complete system. Performance of each layer is mapped to energy by recursively inspecting the lower layers. For brevity, we will sometimes refer to a higher-level as a client of a lower-level one.

We can reason about Energy/Utility with a bidirectional interpretation:

- the (minimal) energy needed for a required utility or
- the (maximal) utility that can be achieved with a given amount of energy.

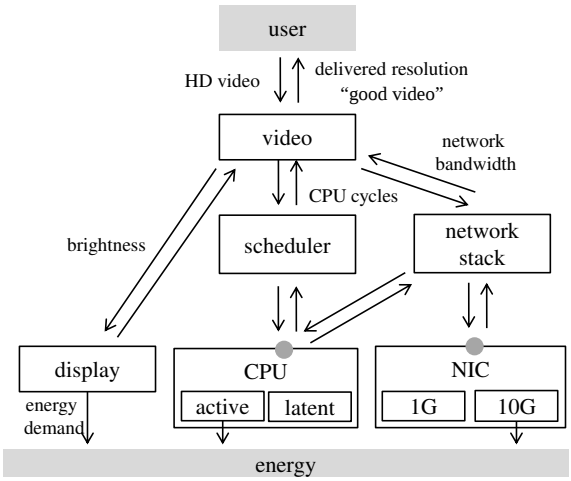


Fig. 3. Hardware software stack for video scenario

For all layers, we use the Energy/Utility ratio for resource management decisions.

Now that we have informally introduced Energy/Utility, we can use the notion with a range of interpretations. For example, consider the scenario in Figure 3 of a network video player requiring three resources, a network connection for streaming, CPU cycles for decoding and a display for showing videos. It is useless to spend all energy on receiving and decoding the video stream in very high quality if the remaining energy does not suffice to display the video in reasonable brightness. In other words, available energy should be used for the resources in order to optimize the usefulness of the video player for its user.

In such cases, where available energy does not suffice to provide all user demands, a rating of the importance of demand to a user is needed. Such ratings can be borrowed from operating-system level mechanisms such as priority or proportional share or from economics expressions of user-level satisfaction.

Examples for utility become closely related to specific applications. These may be QoS parameters for media or the number of transactions per second for database applications. But also more subtle variants of utility are of interest, for example the minimal time required to respond to sudden load peaks. It may be necessary to keep a number of disk drives running to be able to respond faster to load changes than the time it takes to restart drives [1].

For a multi-level scenario to become practical, cross-level global decisions should be restricted to rare decision points and frequent decisions have to be made locally within a level. To enable local and global decisions, energy costs for resources are propagated upwards or required utilities downwards. Consider Figure 1 as an abstract example: applications $A$ and $B$ use an intermediate resource $R_m$ which in turn can use either one of the lower-level resources $R_{l,1}$ or $R_{l,2}$ as in the network example described earlier. Utility of either $R_{l,1}$ or $R_{l,2}$ is used to implement $R_m$, depending on which is more energy efficient. We refer to this type of selection as a local decision between alternative implementations.

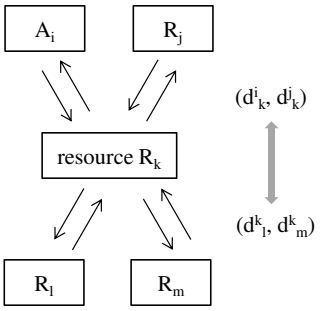


Fig. 4. Translation between higher-level and lower-level demands.

The configuration process can be initiated top down or bottom up. In a top down process, given required utilities for A and B are propagated downwards making the most energy efficient decisions at each level. In a bottom up process, the resource management decisions are guided by the desire to generate the highest possible utility at the application level. This upwards process requires knowledge, how application utilities relate to each other depending on user preference.

## II. Multi-Resource Multi-Level Scheduling

Central to our approach is an efficient translation between the performance requested at the higher level of each resource-layer in the hierarchy and the lower-level demands required to achieve this performance. In general, multiple applications and higher-level resources specify demands for the same shared resource (e.g., $R_k$ in Figure 4) that have to be satisfied simultaneously. For example several applications may share the network and specify demands in terms of latency and bandwidth requirements. The first challenge for determining the low-level demands of such a resource $R_k$ is therefore to combine these demands into a total demand. Depending on the type of resource, this combination may simply be an aggregate (e.g., the accumulated bandwidth or the minimal latency) but there are also resources where these requests have to remain in the form of a vector.

For example, to guarantee a low latency for network traffic, bandwidth must be reserved for this traffic to avoid that low-latency requests are buffered in the switches and elsewhere. An elaborate study of this point is described in the context of data centers in the work of Alizadeh et al. [2]. Their protocols — called HULL — leave bandwidth headroom in switches to guarantee unbuffered transmission of low latency requests. Demands specified to a HULL-enabled network must therefore distinguish whether part of this low-latency bandwidth is used or whether the request falls into the normal high-bandwidth class. The example shows also that demands may be *incompatible*. HULL can serve a combination of high- and low-latency requests up to the bandwidth reservation for both classes. Alternatively requests utilizing the full fabric bandwidth can be satisfied. However, in the latter case, the latency requirements for low-latency requests can no longer be guaranteed. A demand vector with low latency and near fabric bandwidth requests can not be met.

Based on the total demand (which may be a vector of demands), a resource determines what lower-level resources it needs to satisfy this demand and hence achieve the desired

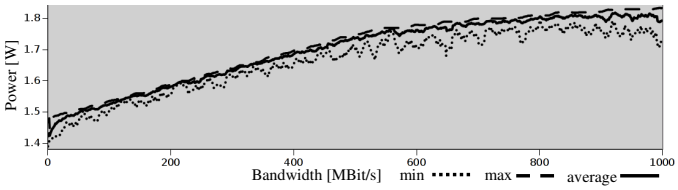


Fig. 5. Energy profile of a 1Gbit Intel EXPI9301CTLK Ethernet card when receiving data at different bandwidths — Source [4].

performance. In the simplest form these requests are singular demands the resource $R_k$ places on the lower-level resources. In Figure 4 these are $R_l$ and $R_m$. We write $d_l^k$ for the demand of $R_k$ to $R_l$ and $d_m^k$ for $R^k$'s demand to $R_m$. In certain situations, it is necessary to expose further information about the relative ranking of these demands to lower level resources. In the context of real-time locking protocols, we see a similar choice between forwarding these "nested" lower-level resource requests with the ceiling priority of the resource or with the current priority inherited from the threads requesting this resource [3].

Once we have translated the user-level demand to a demand for hardware resources, we consider the energy profile of these resources to obtain the power and in turn the energy as the power-time product that this resource requires. Figure 5 contains an example of such an energy profile. It shows the power required by a 1Gbit intel EXPI9301CTLK Ethernet card while receiving data. We measured this power by tapping into the supply lines of a riser card for different bandwidth from idle (0 Mbit/s) to the full capacity (near 1 Gbit/s). It is easy to see that a direct representation of this energy profile is much too detailled to be usable for the rather coarse grain global energy adjustments that we have to make.

Models for other hardware resources have been published [5]–[7]. However, like the profile in Figure 5, these profiles tend to be too detailed for our purpose. In Section IV, we return to this point and give further insights into our approach for reducing this complexity.

There are two primary ways to configure the hardware/software stack of a system.

*Top Down:* Top-down configuration starts with a set of demands that should result in utility and searches as described above for the most energy-efficient configuration that delivers this utility.

*Bottom Up:* Bottom-up configuration starts with an available energy budget and strives to optimize the utility. In principle, all alternative demands of the application are checked in the same way as in top-down configuration and deliver a necessary energy budget. Whichever configuration meets the user's demand best within the available budget is chosen.

## III. Challenges

In addition to the above challenges, for which we will introduce partial solutions in Section IV, we explain also those for which we have no immediate solution at hand.

*1) Complexity of Application Models:* One of the biggest challenges is to obtain the energy profiles and demand/performance functions for all hard- and software resources.

For simple usage characteristics of hardware resources (e.g., for the CPU [8] though not for sophisticated instruction mixed and for an energy-adaptive network interface [4]) we have already found suitable energy profiles (see Section IV) and we are confident to be able to extend these to more complex resources. However, for applications, suitable characterizations in terms of demand/performance functions are rare.

The main difficulty when characterizing more complex applications such as for instance a database management system (DBMS) is to understand the delicate interplay between internal state, frequently changing load and internal optimizations that already try to improve the energy-efficiency of the application. For example, repeated observation of similar requests lets the DBMS cache results (which increases the memory demand of the system) or to switch from a demand driven computation of queries to a setting where a background thread computes part of the queries ahead of time.

We hope that a decomposition of the DBMS into internal schedulers of a multitude of resources reveals an internal structure that is compatible with our approach.

*2) Local Optimization:* Load-based local optimizations *within* a single resource, can provide a significant gain in terms of energy consumption. For example while the network may be configured to use a high bandwidth network card because the increased bandwidth reduces the energy spent in decoding the video, it is still possible to move to a lower bandwidth mode when the current traffic is low enough to warrant such a switch. The components themselves know best when and how to perform such local optimization. Examples of locally optimizing components beside the network are disks that use request reordering and the wireless network with adaptive link rates and transmit energy.

A similar approach to those local optimization can be found in software, which can replace individual sub-components such as search algorithms depending on the current type of work requested. A database is an example that performs such optimization, choosing operator implementations that are the most adequate for a given system load and query composition. This optimization is based on prediction and, if needed, dynamic evaluation of progress and performance during the runtime of a query [9].

While such local optimizations lead to increased energy-efficiency in single components, optimizing a whole system configuration requires a global view. This makes decomposition harder and, with increasing complexity of the software and hardware stack, increases the time required for global reconfiguration and optimization.

The streaming video example introduced in Figure 3 is one example that a global view is needed. The demands requested from the network card in terms of bandwidth may be 2 or 10 Mbit/s depending on the video codec used. We cannot let the network card or even the video player make the decision which bandwidth to use. If we reduce the bandwidth while still retaining the same video quality we will have to use a more

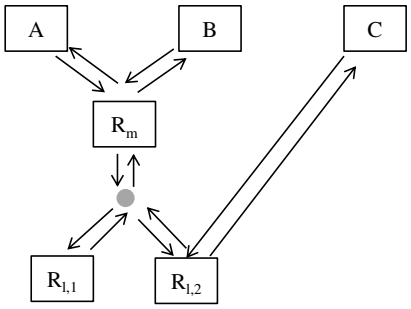


Fig. 6. Choice between alternative utility-generating low-level resources $R_{l,1}$ and $R_{l,2}$ to generate the utility the intermediate resource $R_m$ offers to the applications $A$ and $B$.

sophisticated codec (e.g., MPEG-4 instead of MPEG-2 for a DVB video stream) that is heavier on CPU usage.

Because this trade-off can occur in different parts of the software stack we need a whole system view to see how the decision for lowering the energy-consumption — and thus performance — of one component affects other components.

This means, that a subtree of a system hierarchy cannot be decomposed for the sake of optimization as long as multiple subtrees of the root node contain resources that are traded against each other. In our example CPU and network card are those resources with the video player being the limiting root node.

Figure 6 illustrates another reason why occasional global decisions are needed. Consider that, starting from the scenario from 1, another application C is starting to use $R_{l,2}$. This may cause $R_m$ to become cheaper (e.g., if $R_m$ and $C$ share $R_{l,2}$) or more expensive in terms of required energy (e.g., if both $R_{l,2}$ and $R_{l,1}$ have to be powered) , and the cost associated with $R_m$ is outdated. In that case a complete reorganization of the resource managers is indicated. In real-time systems, such situations are closely related to mode changes and we hope to draw from the insights obtained there.

For our energy aware resource hierarchy we envision components that are *energy-adaptive* on their local level, and *energy-aware* and *configurable* for global system configuration. Components providing these characteristics can enable highly energy-efficient systems.

## IV. Operation Modes: Mastering the Complexity of Energy/Utility-Based Scheduling

In the previous section, we have seen the central role of translating demand and performance between lower and higher levels (Figure 4). We have also seen on the example of the gigabit Ethernet card that hardware profiles, which actually determine how much energy is required for a given system utility, can be rather complex, which would complicate the translation and make our approach intractable. Fortunately, we found that many resources follow a much simpler energy/performance characteristic when we leave fine-grain adjustments to the resource-local schedulers: they either consume energy in nearly discrete steps, which we call *modes* of operation, or in a way that can be approximated as a linear combination of the demands of two adjacent modes.

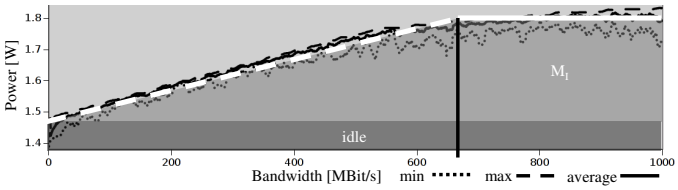


Fig. 7. Modes of a 1 Gbit Intel EXPI9301CTBLK Ethernet card.

Consider Figure 7, which augments Figure 5 in Section II with additional annotations. To keep the card operational without sending or receiving any data, a power of approximately 1.48 W is required. We call this state of operation the *idle mode* of this resource. Once the card receives with a bandwidth higher than approximately 660 Mbit/s, the average power demand is almost constant at 1.8 W. In Hähnel et al. [8] and [4] (from which the above measurement originates), we have seen that the energy profile of many resources show such a flat power demand. For example, the average power demand of a 10 Gbit Intel Ethernet Server Adapter X520-T while receiving at a bandwidth of 10 Gbit/s was only 0.15 W larger than the idle mode power of 7.85 W, which motivated us to characterize the card only with a single operation mode and a constant power demand of 8 W.

In the range between 0 Mbit/s and 660 Mbit/s, the power demand of the 1 Gbit NIC increases nearly linearly from 1.48 W — the idle mode ($M_{idle}$) power — to 1.8 W — the card's single and therefore adjacent non-idle mode which we call $M_I$. If we call these bandwidth values the upper and lower threshold bandwidths for the modes idle and $M_I$ respectively (i.e., the NIC can no longer be idle if a bandwidth greater than $\theta^{up}(M_{idle}) = 0\,\mathrm{Mbit/s}$ has to be received and it starts being in $M_I$ once the bandwidth is greater than $\theta^{low}(M_I) = 660\,\mathrm{Mbit/s}$), we can characterize the power demand $P$ in between these two modes as the linear combination

$$P = \alpha \cdot (b - \theta^{up}(M_{idle})) + P(M_{idle}) \tag{1}$$

In this equation, $b$ is a request (in this case the requested bandwidth) in the interval $[\theta^{up}(M_{idle}), \theta^{low}(M_I)]$. The parameter

$$\alpha = \frac{(P(M_I) - P(M_{idle}))}{\theta^{low}(M_I) - \theta^{up}(M_{idle})}$$

is the gradient of the straight dashed line represented by Equation 1.

Again, in [8], we have seen that this characterization holds for a multitude of resources including the CPU when considering as adjacent modes the idle mode and a full-performance mode at each supported frequency level of the CPU. The full-performance mode was triggered by an artificial benchmark, which we wrote to complete as many integer instructions as possible. Figure 8 shows the measured and estimated power and the relative error of this estimation of 13 SPEC CPU 2006 benchmarks.[1] The estimation accounted halted clock cycles with an idle mode power and used the fraction of non-halted versus halted cycles for the estimation. Please note, all 13

[1] Measurements were taken on an Intel i5-2400S Sandybridge processor running Linux 3.3.4 with no frequency scaling and idle=halt. The measured power for idle and our artificial benchmarks were 7.7 W idle, 10.8 W benchmark at 2.0 GHz, 7.7 W and 12.7 W at 2.5 GHz and 8.1 W and 20.76 W at 3.2 GHz.

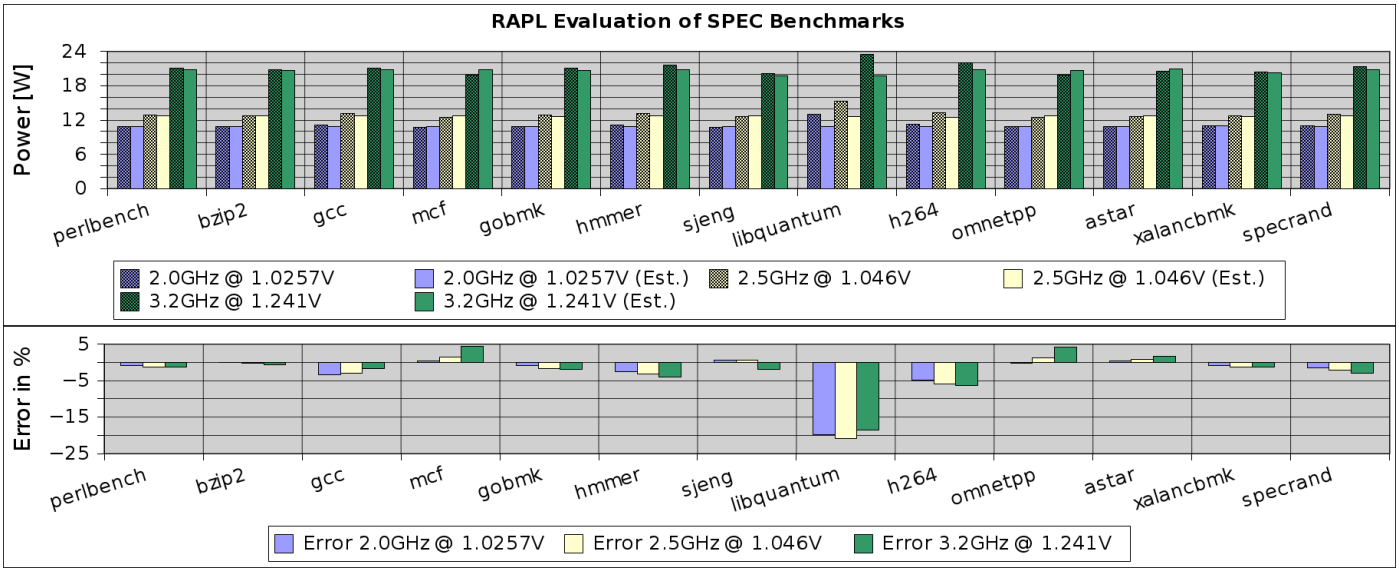


Fig. 8. SPECINT from SPEC CPU 2006 Benchmarks measured using RAPL when estimated by modes Source: [8]

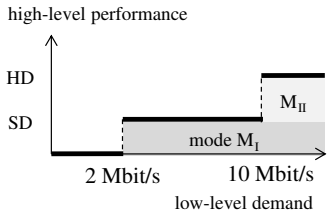


Fig. 9. Mode-based characterization of the video demand/performance function.

benchmarks were SPEC CPU 2006 integer benchmarks (CINT 2006). We expect that for other benchmarks it will be necessary to characterize the floating point unit and other instruction extensions such as Intel AVX or SSE as separate resources with additional modes. Hylick et al. [10] support our findings for disks.

### A. Modes for Software Resources

Before we continue introducing how modes work for software resources let us recall how we are used to construct systems where time is the primary resource to manage. Rather than computing schedules for the actual execution times of tasks, adjusting priorities and other steering metrics based on the actual execution, we compute upper bounds and determine schedulability and resource demands based on these bounds.

To return to our video example, rather than adjusting the network demand to the actual bandwidth required to fetch the next scene, streaming works by allocating a certain amount of memory to buffer part of the video stream before decoding. For a reasonably sized buffer, a network demand of 2 Mbit/s suffices to display an SD video and a demand of 10 Mbit/s sustains the demand for a full HD stream.

Analogous to hardware modes, we therefore strive to characterize demand/performance functions in terms of modes, which describe certain flat levels where the performance is constant despite increasing demand from the considered lower-level resource, or as linear combination of adjacent modes. Figure 9 illustrates these modes in the mapping from video quality to network bandwidth.

### B. Translating Utility to Energy with Modes

With modes, the translation between higher and lower-level demands and ultimately between utility and energy becomes simpler than translating the actual demands down to the lowest level resources and then via complex energy models to the requested energy. Figure 10 demonstrates this translation for our video example and the resource network bandwidth. Using similar demand/performance functions for the resources display and CPU and accumulating the resulting power demands, we obtain the energy required for displaying the video in a certain combination of resource modes as the product of the accumulated power of all resources and of the time that the system remains in this combination of modes.

*Top Down:* The top down translation starts with the demand that the application has to meet (in our case the demand to display a HD video). Going to the right, the smallest lower-level demand that, when satisfied, meets this demand is the low threshold of mode $M_{II}$ with a bandwidth demand of 10 Mbit/s. With this demand, or more generally with the total demand of all applications that need network bandwidth, we can lookup the demand/performance function of the network to find that the demand is above the upper idle mode threshold and below the low threshold $\theta^{low}(M_I)$ of mode $M_I$ of the network. This means, we have to compute the resulting network demand using Equation 1 as a linear combination of the $M_{idle}$ and $M_I$ demands.

Had this total demand been 2 Gbit/s, i.e., above the upper $M_I$ threshold, the 1 Gbit Ethernet card could no longer sustain this traffic. Instead, as described in greater detail in [4], we would have to switch to the 10 Gbit card. As this card, does not show a graceful increase of its energy demand, we have to caculate all further resource requests as if we had requested

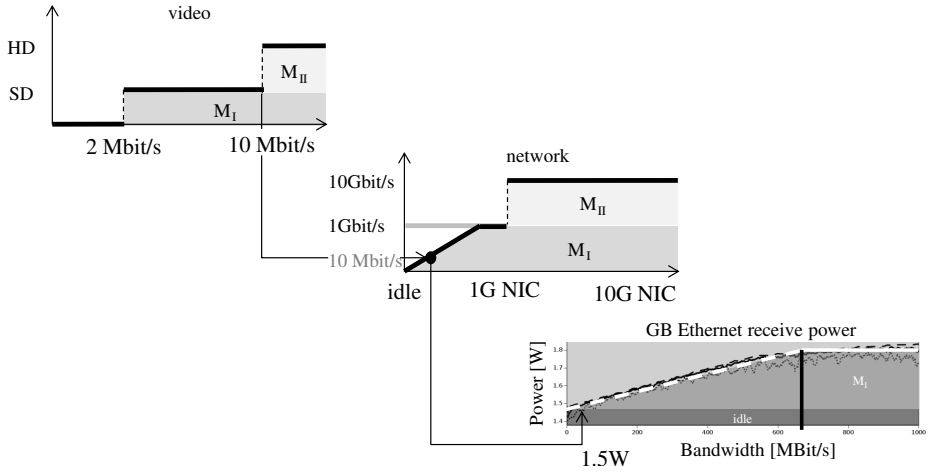


Fig. 10. Mode-based translation between utility and energy.

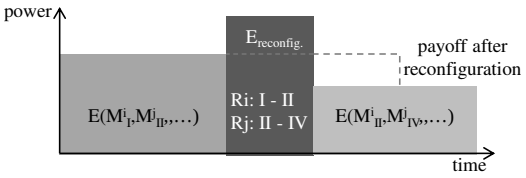


Fig. 11. Change of modes as a result of changes of the workload

10 Gbit/s in the first place (because $\theta^{up}(M_I) = \theta^{low}(M_{II})$ there is no room for a linear combination of $M_I$ and $M_{II}$ and we have to directly switch to $M_{II}$, which corresponds to the 10 Gbit card's only mode of operation.

Fortunately, our network can be serviced with a 10 Mbit/s bandwidth of the 1 Gbit card, which corresponds according to the card's demand/energy function to a power of approximately 1.5 W. In other words, mode $M_{II}$ of the video inflicted the linear combination of $M_{idle}$ and $M_I$ for the network as a whole which in turn inflicted the $M_{idle}$ - $M_I$ linear combination for the gigabit Ethernet card.

*Bottom Up:* The bottom-up translation is as described in Section II except that we expect the characterization of demand/performance functions in modes to reduce the number of possibilities to a degree where the configuration can be transformed into a treatable optimization problem.

*Scheduling based on Modes:* Once we have determined the mode for each resource that best matches the current system-level demand while minimizing the system's overall energy consumption, we have optimally adjusted our system to the current load situation. For as long as this situation persists, the energy demand for this mode combination can be determined from the power demand of all lowest-level hardware resources times the duration how long this situation lasts. Upon a change of requirements (e.g., when the user decides to stop playing the video and switches to a browser) a change of modes may be indicated. Unless this change is short term and local to a few resources, we can simply leave the system in its original mode because the energy to reach a global decision on a new set of modes dominates prospective energy savings.

For more coarse grain fluctuations of the load, such a readjustment is not only justified but may also be required to meet the requested utility. The decision involves finding a new set of modes for all resources and transitioning between these modes. To accommodate for the costs of this mode change, we propose to introduce so-called *transition modes*. The switch from or to such a transition mode can then come for free and the same framework be used to account for these mode switching costs and in particular for the computation required to determine the new configuration. Figure 11 illustrates this mode change and the transition modes $I - II$ for the abstract resource $R_i$ and $II - IV$ for $R_j$. The payoff point is the time, when the gain of operating the resources $R_i$ and $R_j$ in the new mode compensates the energy overhead induced by the reconfiguration

## V. Related Work

We believe this paper to be the first to use Energy/Utility to govern multi-level and multi-resource scheduling as a holistic design principle.

We comment on some related research fields.

*Energy/Utility:* Doug Jensen's seminal work on Time/Utility functions and his frequent lectures in Dresden were one source of our work and certainly an inspiration for the choice of the name.

However, Jensen's time/utility and our Energy/Utility have significantly different semantics, in his own words: "Jensen's time/utility functions (or TUFs) allow the semantics of soft time constraints to be precisely specified. A TUF, which is a generalization of the deadline constraint, specifies the utility to the system resulting from the completion of an activity as a function of its completion time." [11]. In contrast, Energy/Utility specifies the utility that is generated if energy is used to activate resources. Scheduling decisions induced by TUFs — to our understanding — are used for CPU resources and in Ravindran et al. [11] adapted to "CPU scheduling for reduced and bounded system-level energy consumption". Wu et al. [12] introduce a fixed utility-energy ratio (UER) to translate execution time into an energy demand. Analogously

to TUFs, spending energy for a task which completes late results in degraded utility.

The approach advocated in this paper is also related to optimization approaches in model-driven software engineering. For example Götz et al. [13] specify resource mappings for components in contracts and employ integer linear programming (ILP) as an optimization strategy for selecting between alternative implementations of the same interface. In their ILP formulation, alternatives appear as two-value integers denoting whether or not a particular implementation is selected. We are optimistic that we can reuse their or similar work for what we call global optimization in this paper. However, we are convinced that a practically useful design approach requires that a majority of decisions to select a suitable combination must be local.

*Multi-Level Scheduling:*

Multi-level scheduling has found tremendous interest in real-time scheduling theory. Abstractly speaking, it addresses the possibility of scheduling schedulers. As a practical example, CPUs may be scheduled by a hypervisor for virtual machines and each guest running on top of the hypervisor contains schedulers for the virtualized CPUs.

Practically oriented works include proportional-share schedulers for virtual machines, for example in the XEN operating system [14]. Lackorzynski et al. [15] make the point that proportional-share schedulers do not work in systems with mixed-criticality task sets spread over more than one virtual machine. They postulate that in some cases resource requests have to be annotated by higher-level resources such that low-level schedulers can consider their criticality.

In comparison to these contributions, *multi-level* as used in this paper refers to a much simpler hierarchy, namely a hierarchy of resource levels translating between higher-level performance and lower-level demands.

*Multi-Resource scheduling:*

A large portion of classical scheduling work concentrates on scheduling for a single type of resource, namely an active resource like CPU cycles. Relatively little is done for the combined, holistic management of various resources. Examples include Linux-RK aka resource kernels [16], Redline [17] and DROPS [18]. Resource Containers by Banga et al. [19] offer an elegant mechanism for attributing usage to the applications on whose behalf resources are accessed.

These systems consider resources that are instantly available but scarce. Our approach considers energy as prime resource to activate latent resources.

*Energy-Aware Scheduling:*

A further area, which has found tremendous interest in the scheduling theory community, is energy-aware scheduling. Examples includes various forms of frequency- and voltage-scaling based algorithms that describe energy as factor that may vary properties of an active resource(CPU).

In contrast, this paper considers energy as the prime resource to enable various resources.

## VI. Outlook

In this paper we have shown a promising approach to full-system energy optimization using our Energy/Utility concept. We have further discussed some open challenges that remain in the focus of our future work.

So far, we have investigated Energy/Utility only in the context of existing system structures. But, the model presented in Sections I and IV reveals its true potential when applied to analyze the behavior of future system.

For example, in the collaborative research center (CRC) 912 — HAEC, we are currently targeting a highly adaptive hardware architecture called HAEC-Box with many-core chip stacks connected on-board via optical and via off-board wireless links. Figure 12 shows the configuration for $3 \times 3$ chip-stack boards. Antennas in both directions of the board are targeted for 100 Gbit/s interference free links to all chipstacks on the opposing boards. Besides alternative routes via the optical link and the radio beam of neighboring chip stacks, beams can also be combined for increased bandwidth at the cost of interference-induced package losses, giving rise to further adaptation possibilities. We plan to extend our Energy/Utility framework to investigate communication aware scheduling and placement decisions as well as link scheduling algorithms to tune the energy spent for communication to the HAEC-Box's application needs.

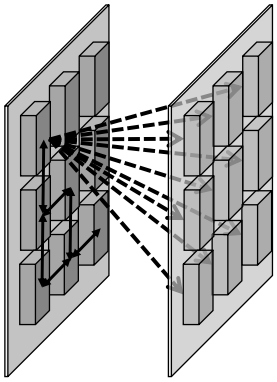

Fig. 12. Chipstacks and communication links of HAEC-Box

## Acknowledgments

This work is in part funded by the DFG through the collaborative research center "Highly Adaptive Energy Efficient Systems" (HAEC) and through the cluster of excellence "Center for Advancing Electronics Dresden" and by the EU and the state Saxony through the ESF young researcher group "IMData".

## References

[1] G. Wang and A. R. B. andi Chris Gniady, "Mitigating disk energy management delays by exploiting peer memory," in *MASCOTS*. IEEE, 2009, pp. 1–4.

[2] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228324

[3] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *Computers, IEEE Transactions on*, vol. 39, no. 9, pp. 1175–1185, 1990.

[4] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "eBond: energy saving in heterogeneous R.A.I.N," in *Proceedings of the fourth international conference on Future energy systems*, ser. e-Energy '13. New York, NY, USA: ACM, 2013, pp. 193–202. [Online]. Available: http://doi.acm.org/10.1145/2487166.2487188

[5] D. C. Snowdon, "OS-Level power management," PhD Thesis, School of Computer Science and Engineering, University of NSW, Sydney 2052, Australia, Mar 2010, available from publications page at http://ssrg.nicta.com.au/.

[6] 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*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 153–168. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966460

[7] Y. Xiao, P. Savolainen, A. Karppanen, M. Siekkinen, and A. Ylä-Jääski, "Practical power modeling of data transmission over 802.11g for wireless applications," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, ser. e-Energy '10. New York, NY, USA: ACM, 2010, pp. 75–84. [Online]. Available: http://doi.acm.org/10.1145/1791314.1791326

[8] M. Hähnel, M. Völp, B. Döbel, and H. Härtig, "The potential of energy/utility-accrual scheduling," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, 2013, pp. 1636–1641.

[9] A. Hameurlain and F. Morvan, "Evolution of query optimization methods," *T. Large-Scale Data- and Knowledge-Centered Systems*, vol. 1, pp. 211–242, 2009.

[10] A. Hylick, R. Sohan, A. Rice, and B. Jones, "An analysis of hard drive energy consumption," in *MASCOTS*, E. L. Miller and C. L. Williamson, Eds. IEEE Computer Society, 2008, pp. 103–112.

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

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

[13] S. Götz, C. Wilke, S. Richly, G. Püschel, and U. Aßmann, "Model-driven self-optimization using integer linear programming and pseudo-boolean optimization," in *Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE)*, 2013, to appear.

[14] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards Real-time Hierarchical Scheduling in Xen," in *Proceedings of the 11th International Conference on Embedded Software*, ser. EMSOFT, Oct. 2011.

[15] A. Lackorzyński, A. Warg, M. Völp, and H. Härtig, "Flattening Hierarchical Scheduling," in *Proceedings of the tenth ACM international conference on Embedded software*, ser. EMSOFT '12, Tampere, Finland, 2012, pp. 93–102. [Online]. Available: http://doi.acm.org/10.1145/2380356.2380376

[16] S. Oikawa and R. Rajkumar, "Linux/rk: A portable resource kernel in linux," in *In 19th IEEE Real-Time Systems Sumposium*, 1998.

[17] T. Yang, T. Liu, E. D. Berger, S. F. Kaplan, and J. E. B. Moss, "Redline: first class support for interactivity in commodity operating systems," in *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 73–86. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855747

[18] H. Härtig, R. Baumgartl, M. Borriss, C.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter, "Drops: Os support for distributed multimedia applications," in *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*, ser. EW 8. New York, NY, USA: ACM, 1998, pp. 203–209. [Online]. Available: http://doi.acm.org/10.1145/319195.319226

[19] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: a new facility for resource management in server systems," in *Proceedings of the third symposium on Operating systems design and implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 45–58. [Online]. Available: http://dl.acm.org/citation.cfm?id=296806.296810