Integrating Fine-Grained Application Adaptation with Global Adaptation for Saving Energy*

Vibhore Vardhan, Daniel Grobe Sachs, Wanghong Yuan, Albert F. Harris,
Sarita V. Adve, Douglas L. Jones, Robin H. Kravets, and Klara Nahrstedt
University of Illinois at Urbana-Champaign
grace@cs.uiuc.edu

Abstract

Energy efficiency has become a primary design criterion for mobile multimedia devices. Prior work has proposed saving energy through coordinated adaptation in multiple system layers, in response to changing application demands and system resources. The scope and frequency of adaptation pose a fundamental conflict in such systems. The Illinois GRACE project addresses this conflict through a hierarchical solution which combines (1) infrequent (expensive) global *adaptation that optimizes energy for all applications in the system and (2) frequent (cheap)* per-application *(or* per-app*) adaptation that optimizes for a single application at a time. This paper demonstrates the benefits of the hierarchical adaptation through a second-generation prototype, GRACE-2. Specifically, it shows that in a network bandwidth constrained environment, per-app application adaptation yields significant energy benefits over and above global adaptation.*

1 Introduction

Mobile devices primarily running soft real-time multimedia applications are becoming an increasingly important computing platform. Such systems are often limited by their battery life, and saving energy is a primary design goal. A widely used energy saving technique is to adapt the system in response to changing application demands and system resources. Researchers have proposed such adaptations in all layers of the system; e.g., hardware, application, operating system, and network. Recent work has demonstrated significant energy benefits in systems that employ coordinated multiple adaptive system layers or cross-layer adaptation [30, 31].

Such systems must employ intelligent control algorithms that determine when and what adaptations to invoke, to exploit the full potential of the underlying adaptations. These algorithms must balance the conflicting demands of adaptation scope and frequency. On one hand, an algorithm that considers all applications and adaptive system layers, referred to as global, is likely to save more energy than a more limited scope algorithm (e.g., considering only one application at a time). On the other hand, global algorithms are also likely to be more expensive since they must optimize across the cross-product of all configurations of all adaptive layers, considering the demands of all (possibly adaptive) applications on these configurations.

Previous cross-layer adaptation work, therefore, performs global adaptation relatively infrequently (e.g., when an application enters or leaves the system [30, 31]). This infrequent invocation in turn reduces the system's responsiveness to change, potentially sacrificing energy benefits. Other work performs adaptations more frequently, but assumes only one application in the system [25] or only a single adaptive layer [8].

To balance the conflict of frequency vs. scope, the Illinois GRACE project (Global Resource Adaptation through CoopEration) takes a *hierarchical approach* that invokes expensive global adaptation occasionally, and inexpensive limited-scope adaptations frequently [24, 30, 31]. GRACE uses three adaptation levels, exploiting the natural frame boundaries in periodic real-time multimedia applications (Figure 1 [24]). *Global* adaptation considers all applications and system layers together, but only occurs at large system changes (e.g., application entry or exit). *Per-application* adaptation (or *per-app*) considers one application at a time and is invoked every frame, adapting all system layers to that application's current demands. *Internal* adaptation adapts only a single system layer (possibly considering several applications) and may be invoked several times per application frame. All adaptation levels are tightly coupled by ensuring that the limited-scope adaptations respect the resource allocations made by global adaptation. The different adaptation levels may or may not consider the same adap-

*This work is supported in part by the National Science Foundation under Grant No. CCR-0205638 and a gift from Texas Instruments.

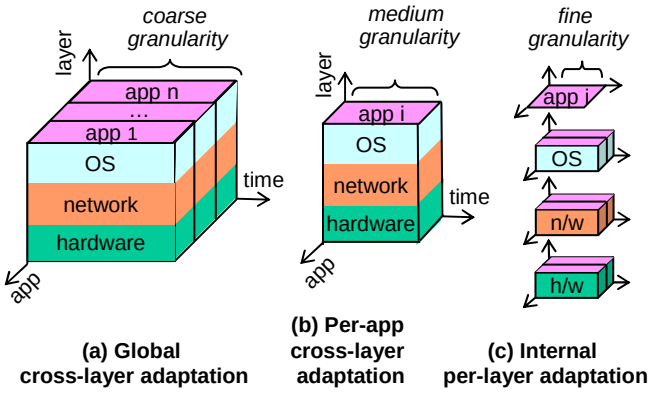

Figure 1. GRACE adaptation hierarchy. (We do not yet adapt the network.)

tations; they are distinguished by the granularity at which they consider an adaptation (e.g., both global and per-app levels may consider dynamic voltage and frequency scaling or DVFS for CPU adaptation).

We previously reported on the first GRACE prototype, GRACE-1, with adaptations in the CPU (DVFS), application (frame rate and dithering), and soft real-time scheduler (CPU time allocation) [30, 31]. GRACE-1's focus was on cross-layer *global* adaptation, for which it showed significant energy benefits. It reported a few experiments with hierarchical adaptation in the CPU and scheduler, but showed only modest benefits over global adaptation

This work focuses on the benefits of hierarchical adaptation in a mobile multimedia system, and reports results from the second generation prototype, GRACE-2. Our main contribution is to show that *per-app application adaptation provides significant benefits over and above global adaptation when network bandwidth is constrained*. These benefits occur with and without per-app CPU adaptation. Notably, the benefits with both per-app application and per-app CPU adaptation are often more than additive. In contrast, GRACE-1 neither provided per-app application adaptation nor implemented a network constraint, and is thus unable to obtain GRACE-2's benefits. Further, GRACE-1's hierarchical adaptation had to be redesigned to incorporate per-app application adaptation because it implicitly assumed a fixed application configuration between global adaptations.

GRACE-2 is implemented on a Pentium M based laptop running Linux 2.6.8-1. As illustrated in Table 1, GRACE-2 implements global adaptations in the CPU, application, and soft real-time scheduler; per-app adaptation in the CPU and application; and internal adaptation in the scheduler. It respects the constraints of CPU utilization and network bandwidth, while minimizing CPU and network transmission energy. All aspects of the system are fully implemented except for network communication. We report both the measured energy savings for the entire system and modeled energy

Objective: Minimize CPU and network transmission energy
Constraints: CPU time, network bandwidth

Layer	Adaptation	Hierarchy level		
		Global	Per-app	Int.
CPU	Dynamic voltage and frequency scaling (DVFS)	yes	yes	no
Application	Drop DCT and motion estimation computations based on adaptive thresholds	yes	yes	no
Scheduler	Change CPU time, network bandwidth budget	yes	no	yes

Table 1. Adaptations supported in GRACE-2

savings for just the CPU and network (we could not isolate the CPU energy through measurements).

We emphasize that the individual adaptations in GRACE-2 are not our focus, and have been previously proposed. Our focus is on their hierarchical control, and specifically on per-app application adaptation.

To our knowledge, this work is the first to demonstrate the benefits (energy savings) from per-app application adaptation over and above global adaptation. It is also the first to demonstrate significant benefits from hierarchical adaptation on a real multimedia system implementing multiple applications, adaptations, and constraints. Section 6 further discusses related work.

2 Layer Adaptations and Models

2.1 CPU

Adaptations: We study dynamic voltage and frequency scaling (DVFS). Our Pentium M CPU supports five frequencies {600, 800, 1000, 1200, 1300 MHz} and corresponding voltages {956, 1260, 1292, 1356, 1388 mV} [13].

To partially alleviate the limitations of the small number of discrete DVFS points supported, we emulate a continuous set of DVFS points as follows [14]. If we need to run at an unsupported frequency, f, we run at the supported frequency just below f (say f_l) for some number of cycles (say c_l) and the supported frequency just above f (say f_h) for the remaining cycles (say c_h). If c cycles need to be executed, then $c_l + c_h = c$ and $\frac{c}{f} = \frac{c_l}{f_l} + \frac{c_h}{f_h}$

Energy model: We report energy measurements from the actual system. However, we could not isolate the CPU energy from the rest of the measured system energy. To better understand the impact of our adaptations on the CPU

Bandwidth (Mbps)	2	5.5	11
Energy per byte (e^{-6} J)	4	2	.08

Table 2. Network bandwidth and energy/byte.

energy and to provide a CPU energy model to the adaptation control algorithms, we use the following: Energy = Power $\times$ Execution Time, where we approximate power at frequency f and voltage V by dynamic power $\propto V^2 \times f$.

We derive the proportionality constant using published numbers for the maximum Pentium M power. The above model does not incorporate leakage (static) power or the effect of application-specific clock gating (as is the case in much of the DVFS literature). These are difficult to incorporate analytically and do not affect the overall trends in the impact of per-app adaptation. This is substantiated by our measured (entire system) energy numbers which do include all effects.

It is noteworthy that CMOS technology is currently in the realm where frequency reductions result in sub-linear voltage reductions. Thus, while previously frequency reductions resulted in quadratic energy reductions (due to linear voltage reductions), this is no longer the case.

2.2 Network (non-adaptive)

We assume a non-adaptive (simulated) network layer with fixed available bandwidth. We model network transmission energy using a fixed energy/byte cost: Network Energy = EnergyPerByte $\times$ BytesTransmitted [4]. Table 2 summarizes energy per byte for different bandwidth values in an IEEE 802.11b wireless network, based on the energy consumption of a Cisco Aironet 350 series PC card [4].

We use different bandwidth values to model different constraints in the system. If the value selected is between two values in Table 2 (possible since not all the bandwidth of the channel is available to one node), we assume transmission cost of the higher bandwidth. We believe our network configurations represent reasonable scenarios seen in practice. Responding to variations in network bandwidth with an adaptive network layer is part of our ongoing work.

2.3 Applications

We consider periodic soft real-time applications or tasks. An application releases a job or a *frame* at the end of each period. We study workloads consisting of various combinations of speech and video encoders and decoders (Section 4). Our H.263 video encoder is adaptive while the other applications are non-adaptive.

Adaptations in the H.263 video encoder: We use the adaptations proposed in [25] (in the context of a system with a single application, and without global adaptation). Since these are not our focus, we only summarize them next and refer to [25] for details.

The adaptations trade off CPU computation (i.e., CPU energy) for the number of bytes transmitted (i.e., network transmission energy), to minimize the total CPU+network transmission energy. The appropriate trade off varies dynamically, depending on the video stream, the system load, and the ratio of network energy per byte to CPU energy per cycle (which depends on the chosen CPU frequency).

The adaptations work at the granularity of a single video frame. They enable dropping certain DCT (discrete cosine transform) computations and motion searches based on a threshold (set by the adaptation control algorithm) for the corresponding frame. The net effect is that, by changing the thresholds, the control algorithm can vary the bit rate and the computation cycles for a frame by about a factor of two. These adaptations can potentially reduce the PSNR (pseudo signal to noise ratio) of the stream, but this is compensated for by adjusting the quantizer step size. Thus, the adaptive encoder can be scaled between a highly compute-intensive but lower bit rate configuration to a less compute-intensive higher bit rate configuration, *without affecting the quality of the decoded video*.

We study four DCT and four motion-search thresholds, resulting in a configuration space of sixteen different encoder configurations.

Deadline misses and frame drops: A frame that does not complete computation or transmission of all its bytes by the end of the ensuing period is said to miss its deadline, with one exception. For video encoders, if a frame finishes its computation within 1ms of its period, we do not count it as a miss. We find these delays do not accumulate (the misses are not clustered). If the video encoder misses its deadline for one frame, the encoding/transmission for that frame continues in the next period, borrowing from the budget of the next frame. If it misses the deadline for two frames in a row, then the next frame is entirely dropped (i.e., incurs no computation or network transmission), enabling the encoder to catch up on its previous frame overruns. We have not (yet) modified the other applications to drop frames.

Since we use soft real-time applications, we assume that we may miss the deadline for or drop a total of up to 5% of all frames, without affecting quality. Although strictly speaking, missing a deadline by a small interval and dropping an entire frame have different effects on quality, we do not distinguish between the two and seek to limit both of these effects to a total of 5%.

2.4 OS Scheduler

We assume an earliest-deadline-first (EDF) soft real-time scheduler for CPU time and network bandwidth. The sched-

uler is responsible for enforcing budget allocations for both CPU time and network bandwidth. To reduce deadline misses due to imperfect predictions of resource demands, the scheduler performs an internal adaptation called budget sharing [2]. Briefly, this allows an application to reclaim unused budget from previous applications' underruns. The EDF CPU scheduler maintains a record of all unused budgets and their expiration times (i.e., the deadline for the job that released the budget). When an application is scheduled, the scheduler first tries to exhaust any unused budget before charging the elapsed cycles to the application. The unused budget can be given to an application only if the expiration time of the budget is less than the deadline for the application [2]. We similarly exploit network bandwidth sharing between applications. Unless stated otherwise, *budget sharing is used in all systems studied here.*

3 Adaptation Control Algorithms

3.1 Global Control

Overview: We use a global control algorithm similar to that in [31], but extended to incorporate a network bandwidth constraint. The algorithm is invoked on large changes in the system; e.g., when an application enters or exits. As input, the algorithm receives the resource requirements (CPU utilization, network bandwidth, CPU+network energy) for each combination of application and CPU configuration. The algorithm must then choose, for each application, the combination of the application and CPU configuration such that (i) the total CPU+network energy is minimized, and (ii) the resource requirements for all the applications (running with the chosen configurations) are met.

More formally, for application i, let Period_i be its period and C_i be a chosen CPU and application configuration combination. Let Energy_{i,C_i} be the energy consumed, Time_{i,C_i} be the CPU time taken, and Bytes_{i,C_i} be the network bytes required by a frame of application i with configuration C_i. Let there be a total of N_{apps} applications in the system and let B be the total network bandwidth (assumed to be fixed). Then the global algorithm must choose the CPU and application configuration C_i for each application i to:

$$\text{minimize} \quad \sum_{i=1}^{N_{apps}} \text{Energy}_{i,C_i}$$

subject to EDF scheduling and bandwidth constraints:

$$\sum_{i=1}^{N_{apps}} \frac{\text{Time}_{i,C_i}}{\text{Period}_i} \leq 1 \quad \text{and} \quad \sum_{i=1}^{N_{apps}} \frac{\text{Bytes}_{i,C_i}}{\text{Period}_i} \leq B$$

Solving the optimization: The above optimization problem is a multi-dimensional multiple-choice knapsack problem (MMKP) [17] and is known to be NP-hard. For the purpose of determining energy savings, we solve this problem using a brute force exhaustive search approach (with one modification below), to give global control the best showing. This approach is impractically expensive for a real system. When reporting the overhead for global, we use a more practical, but possibly sub-optimal heuristic approach based on Lagrangian techniques [17]. (We found the energy savings of both approaches to be comparable for the scenarios studied here.)

To reduce the complexity of both solution approaches, we choose the same frequency (CPU configuration) for all applications. We justify this heuristic by Jensen's inequality [15]: if the CPU energy per unit time is a convex function of frequency, then the best frequency setting is a single point for all applications (if the CPU does not support this single point, then a combination of adjacent supported frequencies is best). This optimization enables us to solve the MMKP problem separately for each supported frequency. We then pick the frequency that provides the minimum energy with the chosen application configurations at that frequency.

After the above process, it is possible that the chosen application configurations and frequency do not exhaust all the CPU utilization and network bandwidth. In that case, the leftover resources are divided among the applications in proportion to their current allocation. This leftover CPU utilization allows a further reduction in frequency. If the resulting frequency is not directly supported, the continuous DVFS emulation discussed in Section 2.1 is used.

Predicting resource requirements: The global algorithm requires predicted resource usage of a frame (Energy_{i,C_i}, Time_{i,C_i}, and Bytes_{i,C_i} in the optimization equations). These predictions must be representative of all frames until the next global adaptation is invoked. Following previous work on resource allocation and scheduling for soft real-time multimedia applications [3, 31], we use profiling of several frames to determine the resource usage. (In our experiments, since our streams are relatively short and since we would like to give global the best showing, we profiled the entire stream off-line.)

To reduce the amount of profiling, we leverage findings from [12]. Specifically, for our applications, the number of execution cycles for a given frame for a given application configuration is roughly independent of frequency; therefore, execution time scales roughly linearly with frequency.[1] Thus, by profiling each application configuration at a single CPU frequency, we are able to estimate the execution time (and the number of bytes) at all frequencies. These estimates also allow estimation of energy using the models in Section 2.

Since we assume a 5% deadline miss rate is acceptable,

[1] This is because these applications generally hit in the cache and do not see much memory stall time [12].

we use the execution time (and bytes) from the frame that falls in the 95th percentile of all profiled frames. For energy, we are concerned with minimization and not meeting a constraint. We therefore use the average time and bytes from the profiled frames as input to the energy models.

In practice, we expect to use on-line profiling of a few hundred frames [29]. For long streams, this poses a negligible overhead. In our experiments, since our streams are short and since we would like to give global the best showing, we profiled the entire stream off-line to determine the 95th percentile and average values.

3.2 Per-App Control

The per-app control algorithm (derived from [25]) is invoked at the start of a frame with the following inputs: (1) the resource allocation for the frame and (2) the resource requirements for the frame for each application configuration. The algorithm then simply chooses the application and CPU configuration combination that has the least energy, and whose CPU time and network bandwidth requirement is within its allocation. If such a combination is not found, then we use the application and CPU configuration of the last frame (likely leading to a deadline miss). The complexity of this algorithm is of the order of the product of the number of application and CPU configurations.

Predicting resource requirements: As for the global algorithm, estimating the execution cycles and bytes for a frame enables estimating all its resource requirements (execution time, bandwidth, and energy). Unlike global control, per-application control requires predicting resource usage for only the next frame.

For non-adaptive applications, we use a common history-based technique, where the average of the execution cycles and bytes in the last five frames is used to predict these quantities for the next frame. For the adaptive application, the history of the past frames may be for different application configurations, and cannot be used directly to predict the behavior of the next frame for yet other configurations. We therefore use an off-line profiling based prediction technique proposed by Sachs et al. as follows [25].

The technique generates an execution cycle predictor off-line by repeatedly encoding one or more sequences (for a fixed hardware frequency), randomly changing the encoder configuration at each frame. This off-line run generates several points for every pair of (previous, next) encoder configurations, mapping the number of cycles in the previous frame to the those in the next frame. The predictor is generated by fitting a function in the least-squared sense, for every pair of (previous, next) configurations. A byte count predictor is similarly generated. To avoid deadline misses, we conservatively add an adaptive leeway into the predicted values for both execution cycles and bytes. Improving the predictors for adaptive applications is part of our future work.

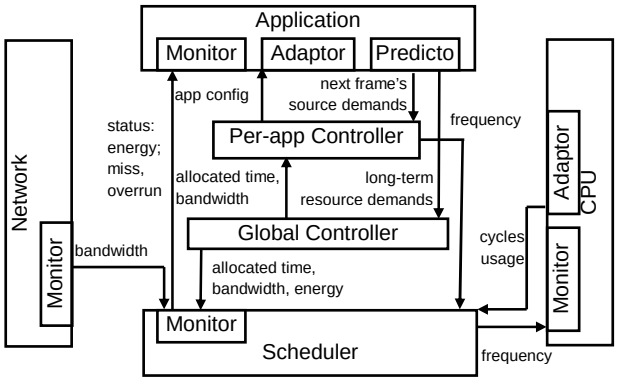

Figure 2. Integrated global and per-application control.

When the per-app adaptation is invoked, it determines the cycle count and byte count for each application configuration for the next frame by using the appropriate predictor, given the knowledge of the previous frame's application configuration, actual cycle count, and actual byte count.

3.3 Integrating Global and Per-App Control

A system that runs with only global control uses the frequency and application configurations as chosen by the global algorithm. In a system that additionally incorporates per-app control, the global algorithm's choice of configuration is only used to determine the resource allocation for each application. This resource allocation is fed as input to the per-app control algorithm. The latter then determines the appropriate configurations for the next frame based on its predictions of the resource usage of that frame and its allocation. Since the per-app controller makes a prediction only for the next frame, based on knowledge of all past frames, it is likely that its prediction is better than that of the global algorithm. Therefore, the per-app controller is likely to better utilize the resources that were allocated to its application by the global algorithm. Figure 2 summarizes the integrated system. As shown, the only interaction between the global and per-app controller is that the former gives the resource allocation to the latter.

4 Experimental Methodology

Implementation: We have implemented all aspects of the system studied except for the network communication (which is replaced with file I/O). Our implementation is on an IBM ThinkPad R40 laptop running the Linux kernel 2.6.8-1, and is described in detail in [27].

Energy measurement: We use an Agilent 66319D sampling power supply to measure the energy consumed by the

#	Applications	Inputs	Fps	Mbps
1	video (enc, enc)	foreman, buggy	30	2
2	video (enc, enc)	foreman, buggy	20	3.3
3	video (enc, dec)	carphone, paris	30	2
4	video (enc, dec) audio (enc, dec)	carphone, paris clinton, lpcqutfe	30 50	2.1
5	video (enc, dec, dec) audio (enc, dec, dec)	foreman, carphone, football female, clinton, male	30 50	6.7

Table 3. Workloads evaluated.

entire system. The measurements were done with the display brightness set to level 3 (0 is minimum). The wireless card was turned off, the laptop battery was removed, and the only applications running were from the experimental workload. All other parts of the system (e.g., hard drive) were on. The network energy used was calculated using the model in Section 2.2, and was added to the above measured energy to give the total system energy in Section 5.3.

Since we cannot isolate the CPU energy in our measurements and since the CPU and the network are the targets of our energy adaptations, our first set of results (Section 5.2) are based on modeled CPU (+network) energy, using the model in Section 2.1.

Workloads: We study various combinations of an H.263 video encoder and decoder and a speech encoder and decoder [28], representing workloads such as remote sensing and teleconferencing [27]. The video encoder is adaptive (Section 2.3) while the other applications are non-adaptive. We use standard video and audio input streams available on the Internet (QCIF size frames for the video encoder and CIF for the video decoder). To study the effect of different types of resource constraints (CPU load, network bandwidth), we run the workloads with different periods (frame rates) for the constituent applications and different values of the available network bandwidth. We studied 16 different workloads covering four resource constraint scenarios: unconstrained, only CPU constrained, only network constrained, and both CPU and network constrained [27]. For space, here we report detailed results from the (most significant) last two scenarios (i.e., those with a network constraint), and summarize the rest. We choose five representative workloads for the network constrained scenarios, summarized in Table 3. Each run includes between 150 to 500 frames for each application.

5 Results

5.1 Overheads

A detailed discussion of experiments showing the overheads of global and per-app adaptation appears in [27], here we briefly summarize our observations. As expected, global adaptation is significantly more expensive than per-app adaptation. For example, in one case, global adaptation took about 4% of a video encoder's average frame computation time, without including the overhead for on-line profiling for predicting the application's resource usage. In contrast, the per-app adaptation overhead was 8X lower. Further, as the number of possible adaptive layers, adaptive components within each layer, and the number of adaptive states within each component increases, the overhead of global will increase much faster than per-app.

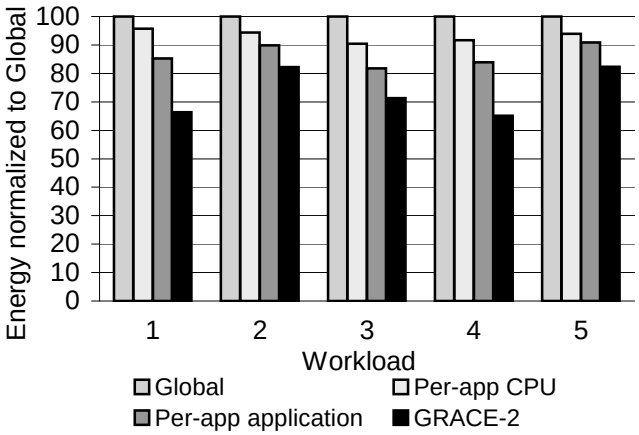

Figure 3. CPU+network energy benefits from per-app application adaptation. For each workload, the leftmost bar shows energy for a system with global adaptation in the CPU, application, and scheduler. The next three bars include this global adaptation as well as per-app CPU adaptation, per-app application adaptation, and both per-app CPU and per-app application adaptation (i.e., GRACE-2) respectively. The energy for each system is normalized to the system with only global adaptation (leftmost bar).

5.2 CPU and Network Energy Savings

Figure 3 illustrates the energy benefits in the CPU-network subsystem of per-app application adaptation. For each workload, the leftmost bar shows a system with global adaptation in the application, CPU, and scheduler. The next three bars shows systems that incorporate this global adaptation and additionally have per-app CPU adaptation (second bar), per-app application adaptation (third bar), and both per-app application and per-app CPU adaptation (the last bar, which represents GRACE-2). The energy of all systems is normalized to that consumed by the system with only global adaptation (the first bar).

Figure 3 shows that adding per-app application adaptation to a system with global adaptation can result in significant energy benefits. The benefits remain significant regardless of whether the base global system contains per-app CPU adaptation (second bar) or not. Relative to a system with only global adaptation, the energy savings from adding per-app application adaptation range from 9% to 18% with an average of 14% for the cases shown here. Relative to a system with both global and per-app CPU adaptation, the energy savings from adding per-app application adaptation range from 12% to 31% with an average of 21%.

It is noteworthy that adding only per-app CPU adaptation to global adaptation gives modest benefits.[2] In contrast, combining CPU and application adaptation at the per-app level gives more than additive benefits in some cases, resulting in quite significant overall savings of hierarchical adaptation relative to a system with only global adaptation.

Experiments from scenarios without a network constraint showed no benefit from per-app application adaptation and modest benefit (6% average) from per-app CPU adaptation, relative to global adaptation [27]. For reference, we also note that global adaptation gave significant benefits (41% average) relative to the non-adaptive system [27].

Analysis: A detailed analysis and supporting data to explain why per-app application adaptation shows significant benefits for the network constrained case and not for other cases appears in [27]. Briefly, for the specific case of our laptop based system, CPU energy dominates over the network energy. Therefore, the application configuration with the least computation is typically the most energy efficient. However, in a network bandwidth constrained scenario, there may be some frames for which this configuration produces too many bytes. Since the global-only system must pick a configuration safe for most frames, it cannot pick this configuration. GRACE-2's frame by frame adaptation, however, is able to pick this configuration for all the frames that produce bytes within the bandwidth constraint, thereby resulting in energy savings.

5.3 System-Wide Energy Savings

We next discuss (measured) system-wide energy savings of GRACE-2 over a system with only global adaptation.[3] Across all workloads in the scenarios with a network constraint reported in [27], we found that GRACE-2's per-app adaptation provides a system-wide energy benefit of 7% to 14% with an average of 10% (relative to only global adaptation). These savings are significant, considering that they are for the entire system including the display, disk, power-supply loss, and memory system; they are actual measured values; and they come from only adaptation of the CPU and application. (As reference, the one workload with multiple applications reported for GRACE-1 showed system-wide savings from hierarchical adaptation of only 3.8%, relative to global adaptation [30].)

[2] The benefits from CPU adaptation are modest relative to those seen for DVFS in much prior work due to the sub-linear relationship between frequency and voltage reductions in recent processors (Section 2.1).

[3] As explained, the network energy is realistically modeled, but is a very small part of the system energy [27].

5.4 Deadline Misses and Budget Sharing

The main benefit of budget sharing (i.e., the internal scheduler adaptation described in Section 2.4) is in reducing the number of deadline misses (including frame drops). Budget sharing has negligible ($< 1\%$) effect on energy. GRACE-2 shows acceptable deadline misses (within 5%) for each application in each scenario/workload studied. Without budget sharing, the deadline miss ratios are high (up to 23%) for several cases. Thus, budget sharing is effective and critical for our system.

6 Related Work

There has been a large amount of work on energy and bandwidth driven adaptations and resource allocation that is relevant to this work. This includes CPU adaptation with and without coordination with a real-time scheduler (e.g., [1, 8, 19, 20, 22, 26, 32]), adaptation of one or more applications with and without OS/middleware support (e.g., [6, 7, 9, 10, 16, 18, 21]), and single-layer or cross-layer adaptation or resource allocation with only global control supporting multiple applications (e.g., [11, 33, 23]) or only per-app control supporting a single application (e.g., [25]). The focus of this work, however, is on hierarchical adaptation control in a cross-layer adaptive system, and more specifically on fine-grained (per-app) application adaptation. None of the above systems exhibit this property.

The systems most closely related to the hierarchical adaptation of GRACE-2 are GRACE-1 [30, 31] which has already been discussed and Fugue [5]. Fugue proposed adaptation at multiple time scales for wireless video [5]. This is one of the key features of GRACE-2's hierarchical control. However, Fugue differs from GRACE-2 in the following important ways. First, it considers only one application running. Second, it is based on the insight that different types of adaptations work on different time scales; e.g., application quality control must occur at a coarser time scale than network transmission power control. GRACE-2's global and per-app controllers consider the same set of adaptations, but for different purposes – the former uses them for resource allocation among multiple applications while the latter does the actual adaptation. Incorporating adaptations that inherently work at different time scales can

be viewed as an orthogonal issue – our system incorporates these as well, but that is not the focus of this work.

7 Conclusions

The GRACE project balances the scope and frequency of energy saving adaptations in multiple layers through a hierarchical approach, where expensive and infrequent global adaptation allocates resources among applications based on long-term predictions, and inexpensive per-application control seeks to make the energy-optimal use of these resources through localized short-term predictions and cross-layer adaptations.

This paper presents results from the second generation prototype, GRACE-2. Specifically, it shows that per-app application adaptation provides significant benefits over and above global adaptation when the network bandwidth is constrained. These benefits are seen both with and without per-app CPU adaptation. For example, the energy savings in the CPU+network from adding per-app application adaptation to a system with global adaptation and per-app CPU adaptation were seen to be up to 31% (average 23%). Interestingly, when both per-app CPU and per-app application adaptation are added to a system with global adaptation, the combined benefits are more than additive.

To our knowledge, this work is the first to demonstrate the benefits from per-app application adaptation control over and above global control. It is also the first to demonstrate significant benefits from hierarchical adaptation on a real multimedia system implementing multiple applications, adaptations, and constraints. Given the low overhead of per-app control and the relatively low added system implementation complexity over a system with global control, the benefits achieved seem worthwhile to exploit.

Our ongoing work is incorporating an adaptive network layer that responds to variations in network bandwidth, and is also exploring other possible application adaptations including those that affect user perception.

References

[1] H. Aydin et al. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *RTSS*, 2001.

[2] M. Caccamo et al. Capacity sharing for overrun control. In *RTSS*, Dec. 2000.

[3] H. H. Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *ICMCS*, 1999.

[4] Cisco Aironet 350 Series Client Adapters Datasheet. http://www.cisco.com/en/US/products/hw/wireless/ps4555/products_data_sheet09186a0080088828.html, 2004.

[5] M. Corner et al. Fugue: time scales of adaptation in mobile video. In *MMCN*, Jan. 2001.

[6] E. de Lara et al. HATS: hierarchical adaptive transmission scheduling for multi-application adaptation. In *MMCN*, 2002.

[7] C. Efstratiou et al. A platform supporting coordinated adaptation in mobile systems. In *WMCSA*, June 2003.

[8] K. Flautner and T. Mudge. Vertigo: Automatic performance-setting for linux. In *OSDI*, Dec. 2002.

[9] J. Flinn et al. Reducing the energy usage of office applications. In *Proc. of Middleware*, Nov. 2001.

[10] J. Flinn and M. Satyanarayanan. PowerScope: A tool for proling the energy usage of mobile applications. In *WMCSA*, 1999.

[11] K. Gopalan and T. Chiueh. Multi-resource allocation and scheduling for periodic soft real-time applications. In *MMCN*, Jan. 2002.

[12] C. J. Hughes et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *Proc. of the 28th Annual Intl. Symp. on Comp. Architecture*, 2001.

[13] Intel Pentium M Processor Datasheet. http://www.intel.com/design/mobile/datashts/25261203.pdf, 2003.

[14] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISLPED*, 1998.

[15] S. Krantz, S. Kress, and R. Kress. *Jensen's Inequality*. Birkhauser, 1999.

[16] M. Mesarina and Y. Turner. Reduced energy decoding of MPEG streams. In *MMCN*, Jan. 2002.

[17] M. Moser et al. An algorithm for the multidimensional multiple-choice knapsack problem. In *IEICE Trans. Fundamentals of Electronics*, 1997.

[18] B. Noble et al. Agile application-aware adaptation for mobility. In *SOSP*, Dec. 1997.

[19] T. Pering et al. Voltage scheduling in the lpARM microprocessor system. In *ISLPED*, July 2000.

[20] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, 2001.

[21] C. Poellabauer et al. Cooperative run-time management of adaptive applications and distributed resources. In *Proc. 10th ACM Multimedia Conf.*, Dec. 2002.

[22] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *DAC*, 2001.

[23] C. Rusu et al. Maximizing the system value while satisfying time and energy constraints. In *RTSS*, Dec. 2002.

[24] Sachs et al. GRACE: A Cross-Layer Adaptation Framework for Saving Energy. *SIDEBAR in IEEE Computer*, dec 2003.

[25] D. Sachs et al. Adaptive video encoding to reduce energy on general-purpose processors. In *ICIP*, Sept. 2003.

[26] T. Simunic et al. Dynamic voltage scaling and power management for portable systems. In *DAC*, 2001.

[27] V. Vardhan et al. Integrating Fine-Grained Application Adaptation with Global Adaptation for Saving Energy (extended version). Technical report, UIUC, 2005. http://www.cs.uiuc.edu/grace/papers/perapp-tr.pdf.

[28] Xiph.org. Speex. http://www.speex.org/, 2003.

[29] W. Yuan. GRACE-OS: An Energy-Efficient Mobile Multimedia Operating System. PhD thesis, UIUC, 2004.

[30] W. Yuan et al. GRACE: Cross-Layer Adaptation for Multimedia Quality and Battery Energy. *IEEE Trans. Mobile Computing*. Accepted for publication.

[31] W. Yuan et al. Design and evaluation of cross-layer adaptation framework for mobile multimedia systems. In *MMCN*, 2003.

[32] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *SOSP*, 2003.

[33] H. Zeng et al. Ecosystem: Managing enegy as a first class operating system resource. In *ASPLOS-X*, 2002.