GRACE: A Hierarchical Adaptation Framework for Saving Energy [*]

Daniel Grobe Sachs*, Wanghong Yuan†, Christopher J. Hughes†, Albert Harris†
Sarita V. Adve†, Douglas L. Jones*, Robin H. Kravets†, Klara Nahrstedt†
∗ Electrical and Computer Engineering
† Department of Computer Science
University of Illinois at Urbana-Champaign
grace@cs.uiuc.edu

Abstract

Mobile systems primarily processing multimedia data are expected to become important platforms for pervasive computing. These systems, however, must satisfy large, dynamic demands of multimedia applications subject to stringent energy, computational, and bandwidth constraints. At the same time, multimedia applications provide the possibility of *adaptation*, allowing tradeoffs between energy, computation, and network bandwidth to maximize the user's experience for the current resources.

Researchers have proposed adaptations in the hardware, network, operating system, and applications to provide QoS guarantees and to save energy. To reap the full benefits of such adaptations, however, the different system layers and applications must coordinate their adaptations with each other. This paper describes a framework, called *GRACE*, to achieve such a coordination, using a novel hierarchical approach that combines global, per-application, and per-layer internal adaptation, for multimedia applications running on wireless systems. GRACE achieves the benefits of coordination through cleanly defined interfaces that keep the internals of the different layers isolated from each other. Our results so far show the effectiveness of the hierarchical adaptations, and justify the use of coordinated, cross-layer adaptations to both save energy and improve the user's multimedia experience.

1 Introduction

Wireless mobile devices, primarily processing multimedia data such as video, audio, and images, are becoming important platforms for pervasive computing. Compared to conventional desktop and server systems, such mobile systems bring new challenges for two reasons: (1) multimedia applications present dynamically changing computation and communication requirements that must be met in soft real-time, and (2) system resources (such as CPU time, network bandwidth, and battery energy) are limited and also dynamically change over time. Thus, mobile systems need to support application quality of service (QoS) in the presence of multiple dynamic resource constraints and dynamic application requirements.

At the same time, our target mobile systems also introduce new opportunities. First, unlike best-effort applications, multimedia applications often result in some computational slack. For example, from the perceptual quality point of view, there is no benefit to completing the decoding of a video frame before the next frame is available. Such slack can be exploited by slowing down hardware resources to save energy, without affecting application quality. Second, multimedia applications can support multiple quality configurations, trading off the quality for resource demands. For example, a video player can decode video frames in different frame size and resolutions based on currently available resources.

Based on the above observations, researchers have introduced adaptive techniques within applications [2] and within various system layers, such as the hardware [16], OS [10], network protocols [7], and middleware [5]. These adaptations have been shown to be effective both for QoS provisioning and for energy saving. However, most of the prior adaptation work focuses on adapting a single layer at a time (possibly in response to changes in another layer). Such independent adaptations in multiple layers could potentially conflict with each other or miss system-wide optimization opportunities.

This paper describes the Illinois GRACE (Global Resource Adaptation through CoopEration) project, which is a cross-layer adaptation framework for saving energy in mobile multimedia systems. All system layers and applications are allowed to be adaptive. These adaptive entities cooperate with each other to achieve a system-wide optimal configuration (i.e., maximize system utility) in the presence of changes in the available resources or application demands.

Our framework currently considers the resources of CPU time, network bandwidth, and the CPU and network transmission energy. It considers adaptations in the hardware layer for the CPU (e.g., voltage and frequency scaling and architecture adaptations), network layer (e.g., adapting transmission power, changing to active, idle, or sleep states of the wireless card, alternating between ARQ, FEC, or a hybrid for reliability), the CPU scheduler (adapting CPU time budgets), the network scheduler (adapting bandwidth budgets), and multimedia applications (e.g., changing video compression amounts to trade off communication and computation without affecting user utility,

[*] This work is supported in part by an equipment donation from AMD, a gift from Motorola Inc., and the National Science Foundation under Grant No.CCR-0096126, CCR-0209198, CCR-0205638, and EIA-0224453. Sarita Adve is also supported by an Alfred P. Sloan Research Fellowship and Christopher J. Hughes was supported by an Intel graduate fellowship.

or changing frame size to trade off utility and all resource use).

Key challenges in designing such a cross-layer adaptive system are: (1) achieving the benefits of continuous global cross-layer adaptation with low overhead, (2) predicting the resource usage of adaptive applications on adaptive system configurations sufficiently well and enough in advance to trigger the right adaptations, and (3) choosing which of possibly tens to thousands of configurations to use given the resource predictions. Finally, although we desire that all system layers and applications coordinate their adaptations, we also desire to preserve the isolation and independence of different layers for practical considerations such as ease of development and maintenance.

GRACE adopts a novel hierarchical approach with three levels of adaptation to address several of the above challenges. The first level is global adaptation, which considers the entire system while adapting, but cannot occur frequently enough due to its inherently high overhead. The other two adaptation levels are limited in scope (per-application and per-layer internal); these are lower overhead and hence can occur more frequently to exploit short-term variations. To ensure that the limited-scope adaptations do not subvert the coordination achieved by the global adaptation, all adaptation levels are tightly coupled.

GRACE is also designed so that no application or system layer need expose its internals to other parts of the system. This is done by carefully defining the interfaces among different layers; these interfaces only encode the external impact of different configurations, but do not divulge how that impact is achieved.

The GRACE framework leverages prior adaptation techniques in all of the system layers we consider (e.g., [16, 10, 7, 5]), and shares some similarity with other coordination frameworks (e.g., [9, 1, 18, 12, 4, 11, 3]). GRACE distinguishes itself from prior work for two reasons: (1) It considers adaptations in all system layers, from the hardware to applications. (2) It employs a hierarchy of adaptations to achieve high responsiveness with small overhead. For example, Q-RAM [9], HATS [3], and Q-fabric [11] consider adaptations in OS resource management and applications only, while the approaches in [18, 12] adapt the hardware and OS only. Our previous work has reported experience with prototypes focusing on parts of the design decisions in GRACE [17, 14, 13]; this paper focuses on describing the full GRACE framework in detail and is the first to describe all three levels of adaptation.

2 The GRACE Framework

An ideal cross-layer adaptive system would continuously monitor each application's resource demands and available resources. At any change, it would consider each possible combination of configurations of the different system layers and applications as a candidate for the next instant. It would choose the combination that maximizes utility while meeting application demands within the available resources. The optimal choice would require perfect knowledge of future resource demands and availability. In practice, however, such an ideal system is infeasible since continuous global adaptation would incur unacceptably large overhead, and it is difficult to have perfect knowledge of the future.

We identify three key challenges that must be resolved by a practical cross-layer adaptive system to approach the above ideal:

- How to get the benefit of continuous global adaptation, but with acceptable overhead?
- How to predict future resource demands and availability, especially given that the configurations for the applications and system layers will change?
- How to choose an optimal configuration, given the above predictions?

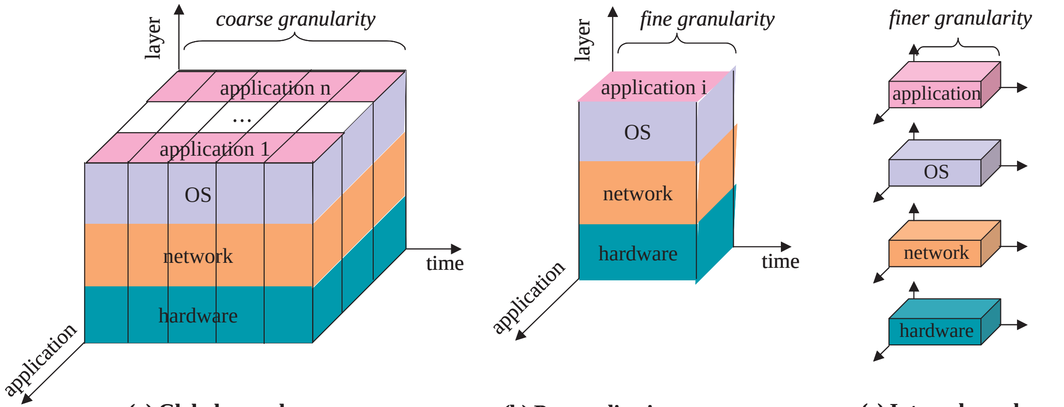
Sections 2.1, 2.2, and 2.3 describe how GRACE meets the above challenges. Section 2.4 puts these parts together to summarize the operation of the entire GRACE framework. The following assumes that multimedia applications are soft real-time and periodic. They release a *job* (e.g., frame decoding) every period, with a soft deadline as the end of the period. We use *utility* to quantify the quality of each application configuration; e.g., based on PSNR (peak signal-to-noise ratio) for video or other basic quality models [2] or user preferences [8]. *System utility* is the weighted sum of utilities of all concurrent applications and measures the overall quality. Further, the OS provides soft real-time CPU and network schedulers which constrain the resource allocation. For example, the earliest deadline first or EDF CPU scheduling algorithm requires that the sum of CPU utilizations of all applications is no more than 1, providing the CPU time constraint for the system.

2.1 Hierarchical Adaptation – Balancing Scope and Temporal Granularity

The ideal system described above performs adaptations that are global in scope (i.e., considers all applications and system layers) and occur frequently (i.e., in response to any change at any time in the system). Unfortunately, adaptations with global scope potentially incur large overheads which makes it impractical to invoke them frequently. Consider, for example, that each application or system layer may have tens to thousands of possible configurations. A global adaptor would need to predict the resource demands for each application configuration while running on each system configuration, and then choose the combination that satisfied the current resource constraints with the maximum utility. Exploring the full, or even partial, cross-product of all of the configurations is expensive, both in terms of the raw computation required and the communication that would be incurred across different layers (e.g., messages and context switches crossing application boundaries, system libraries, and the kernel).

On the other hand, performing global adaptation too infrequently risks inadequate response to intervening changes, and not performing global adpatation at all risks poor or even conflicting configuration choices in the independent layers.

Thus, our first challenge is to design a system with an adequate balance between the scope and the temporal granularity of adaptation. We use a *hierarchical* approach to solve this problem: GRACE performs expensive global adaptations occasionally, and limited-scope but inexpensive adaptations constantly. The combination of the different levels of adaptation allows us to achieve most of the benefits of continuous,



(a) Global cross-layer adaptation

(b) Per-application cross-layer adaptation

(c) Internal, per-layer adaptation

Figure 1: Hierarchical adaptation. GRACE uses three levels of adaptation differing in scope and temporal granularity: (a) *global* – adapts all system layers and applications, invoked infrequently for large changes, (b) *per-application* – adapts all system layers for one application, invoked once per application job, (c) *internal* – adapts a single system layer or application (the system layer adaptation could apply to multiple applications), invoked frequently (e.g., every packet in the network, every few instructions in the hardware, every scheduling slice in the scheduler).

global adaptations without incurring the overhead of running full cross-layer adaptations.

GRACE identifies and supports three levels of adaptation, which increasingly trade off scope (and thus overhead) for finer temporal granularity, illustrated in Figure 1:

- *Global* adaptation considers all applications and all system layers together. It is invoked infrequently, in response to large changes in resource demands and/or availability (e.g., when an application joins or leaves the system).
- *Per-application* adaptation considers only one application at a time, and adapts that application and all system layers for that application. It is invoked at the start of each job of the application.
- *Internal* adaptation adapts only a single system layer or application, but the system layer adaptation could apply to multiple applications. It may be invoked at a granularity finer than a job (e.g., every packet in the network, every few hundred instructions in hardware, or every scheduling slice in the scheduler).

To ensure that the limited-scope per-application and internal adaptations do not subvert the coordinated full-system decisions made by global adaptation, all three adaptation levels are tightly coupled with each other. In particular, the global adaptation determines a utility and relative resource allocation (CPU time, network bandwidth, and energy) for each application that maximizes overall system utility. This determination takes into consideration the impact of any limited-scope adaptations. The limited-scope adaptations, in turn, ensure that they respect the globally assigned utility and resource allocations over the scope of their adaptations.

In combination, the three levels of adaptation are able to respond to all types of changes in resource demands and resource availability, while respecting the cross-layer coordination achieved by the global adaptation. The three adaptation levels are described in more detail next.

Global adaptation: The goal of global adaptation is to allocate available resources (CPU time, network bandwidth, and energy) among all applications in a way that will maximize system utility.

A *global coordinator* performs the resource allocation by searching through the space of all combinations of configurations of the different system layers and applications. It determines the overall utility and resource usage for each such combination. The combination that maximizes the utility without exceeding the available resources is chosen as the desired set of configurations. This choice defines for each application the utility it must achieve (i.e., the utility corresponding to the chosen configuration for the application) and its relative resource allocation (i.e., the time, bandwidth, and energy used by that application configuration on the chosen configurations for the CPU and network). Subsequent levels of adaptation view this assignment of utility and resource allocation as a contract they must respect.

The global adaptation process involves making predictions of resource demands and availability, and choosing the best configuration. Sections 2.2 and 2.3 respectively discuss techniques for these purposes. While these techniques are practical, they are still complex. Global adaptation is therefore invoked infrequently, only in response to large changes in the system (e.g., when an application enters or leaves the system, when the resource usage of an application changes significantly due to changes in the media stream, or when the available network bandwidth or system energy drops significantly).

The long time interval between global adaptations, however, implies its configuration choices could be sub-optimal. This is because the underlying resource predictions also need to be made for the long time interval, and hence are necessarily rough estimates based on average statistical behavior. The consequent configuration choices may not be optimal for short-term variations that may occur during this time interval. The per-application and internal adaptations compensate for this suboptimality as discussed next.

Per-application adaptation: Per-application adaptation is invoked at the start of each job of an application. At this time, there is more accurate information about the job's resource demands than available to the global adaptation (e.g., the resource demand for a job is well correlated to the demand exhibited by the last few jobs, and can be derived by maintaining limited history). Similarly, information on currently available resources is also more accurate. Thus, at this time, a more informed choice can be made for the job. However, the overhead of determining a system-wide optimal configuration is too much to pay at the start of each job. Therefore, the per-application adaptation does not attempt to reallocate resources among different applications, but continues to use the same allocation (with an exception described below).

The goal of the per-application adaptation is to find the application and system configuration that will provide the utility expected by the global coordinator, within its CPU time and bandwidth budget allocation, with minimal energy.

The per-application adaptation makes a distinction between the resources of CPU time and network bandwidth, and energy. The former two are not conservable resources (i.e., if CPU time and network bandwidth go unused, they cannot be conserved for the future), while the latter is and can be saved for later. Given that future short-term variations in resource demands cannot be predicted, the per-application coordinator seeks to minimize its use of energy while ensuring that it uses up the time and bandwidth allocated. If less energy is used than that allocated by the global coordinator, it can be "banked" for later jobs which could be longer than the current one.

It may be possible for the per-application coordinator to find configurations that increase the utility for the next job; however, we do not allow this since it could potentially introduce rapid fluctuations in the quality of the multimedia stream which could be annoying to the user.

Finally, as described below, internal scheduler adaptations can temporarily affect the CPU time available to a job (e.g., more time may be available if a previous job of another application finished early). The per-application adaptation also incorporates any such temporary resource availability changes in its optimization process (if so indicated by the scheduler).

Internal adaptation: Internal adaptation adapts only a single system layer or a single application at a time (a system layer adaptation could apply to multiple applications, however). Since internal adaptation does not need to consider a cross-product of configurations of different layers and/or applications, it is significantly more efficient and can potentially be invoked at a very fine temporal granularity.

Internal adaptation is useful in many ways. For example, recall that time and bandwidth allocations are made globally assuming an average resource demand for each job. However, a given job of an application may underrun or overrun this demand, and the wireless channel quality may change temporarily. In response to such variations, the CPU and network schedulers can adapt by redistributing the allocated time and bandwidth more optimally.

Second, internal adaptation can respond to variations in resource usage and resource availability that occur *within* a given job. For example, different parts of the job may use different CPU resources. Under-utilized resources could be deactivated, thereby saving energy but without affecting CPU time. Similarly, rapidly changing characteristics of the wireless channel motivate network layer adaptations at the packet granularity, to achieve a given net bandwidth and quality for an application.

Third, during the process of global and per-application adaptation, a system layer may apply an internal adaptation process to determine its minimal energy configuration, given an application configuration and resource allocation (motivated further in Section 2.3). This can cull a significant part of the full cross-product configuration space that must be searched by the global or per-application adaptation coordinators. In particular, it allows each system layer to locally integrate the effect of any intra-job internal adaptations – exposing those adaptations to the global or per-application coordinators would significantly explode the configuration space and require the coordinators to know too much about the internals of the individual layers.

As mentioned, there has been much work in adapting a single layer or application at a time, and we can leverage all of that work for internal adaptations. We therefore focus the rest of this paper on global and per-application adaptations.

2.2 Predicting Resource Demands

All the adaptation levels require predicting the resource demands for each application. The global level requires a long-term prediction while the per-application adaptation requires a prediction for the next job. Current real-time schedulers also need to make long-term resource usage predictions. For example, hard real-time CPU schedulers use a combination of measurement and analysis to determine the worst-case execution time of a job and schedule accordingly. Soft real-time schedulers often profile (i.e., run) several jobs and use the measured average or some other statistic (e.g., 90th percentile) for job CPU time. Programmer input could also be used for this purpose. For short-term predictions, a common technique is dynamic history-based predictors. For example, the CPU time for the next job may be predicted as the average (or maximum or some other statistic) of the CPU time of the last N jobs (where N is a small number).

Using the above techniques in the context of GRACE is not straightforward since resource usage in GRACE depends on the specific system and application configuration in use. For example, we can using profiling to determine the long-term average per-job CPU time, but the profiles would need to be collected for each combination of application and system configuration, with each profile run over a large number of frames. The problem with the short-term predictions is even more acute. It is hard to develop history-based predictors because the previous N jobs may have run with different system and application configurations. Correlating the CPU time (or other resource usage) of those runs with that for the next job is difficult. Even if it were possible to form such correlations, it would require deep knowledge of the different application and system layer configurations, which is impractical and undesirable to incorporate within a single centralized entity.

Our approach to reduce the complexity of the prediction problem is to divide it into two parts – a system configuration independent part that can be predicted entirely by the application, and a system configuration dependent part that is handled by the specific system layer. For example, as applied to the CPU [15], the CPU time demand of a job can be divided into the number of instructions in the job and time per instruction (or TPI, which is determined by the product of CPU frequency and instructions-per-cycle or IPC). The number of instructions in a job is entirely an application-level parameter, independent of the rest of the system configuration. Its prediction can therefore be localized within an application-level predictor, oblivious to the rest of the system. The time-per-instruction or TPI prediction can be made within the hardware layer. Previous work has shown that, for several multimedia applications, TPI for a job stays roughly constant across all jobs of the same type[1] for a given CPU and application configuration [6].[2] TPI (effectively IPC) can therefore be measured by profiling a job for each application configuration on each system configuration once at the beginning of the application run. Analogous observations hold for CPU energy, which can be divided into instructions and energy per instruction.

Similarly, to quantify network bandwidth and energy demands in system-independent terms, we use the *application-level traffic* (number of bytes generated by the job) to be transmitted per job and any application specified QoS requirements (e.g., acceptable loss rate). The application-level traffic is independent of the network layer protocols, and again can be estimated using a purely application-level predictor. Depending on the network protocol used (e.g., use of ARQ vs. FEC vs. some hybrid for reliability) and the current channel quality, more bytes will be added per job or the transmission power may be increased, affecting the total expected bandwidth and transmission energy. However, the impact of this can be determined purely in the network layer, and does not need to be made visible to the application. Analogous observations hold for the additional CPU time and energy spent by network processing on behalf of the application.

In summary, with each application, we associate a predictor of the application's resource demands in system-independent terms. With each system layer, we associate a resource demand predictor that takes the system-independent estimate for an application configuration and converts it to an absolute measure for the resource demand (time, energy, and/or bandwidth) for a specific configuration of that system layer. These predictors form the key interface between the application and system layers (the actual communication between them is orchestrated by the operating system as described later). The predictors could be implemented by the designers/vendors of the adaptive application or system layer, as a user-level library or as part of the operating system. Further, our approach does not preclude non-adaptive applications and system layers – the "predictors" associated with these default to the conventional techniques currently used as described earlier.

2.3 The Optimization Process

The third challenge for GRACE is to develop, for each adaptation level, an efficient optimization process that picks the best system and application configurations. Our approach exploits the hierarchical adaptation framework and the ability to express resource demands in system-independent parameters.

Per-application optimization. The per-application coordinator must determine the best configuration for its application and the other system layers that minimizes energy, given the resource allocation and utility assignment for the application (obtained from the other adaptation levels). For simplicity, below, we assume that the CPU and network are independent system layers; i.e., network protocol processing occurs on a separate hardware. The optimization process is as follows:

1. The per-application coordinator first queries the application predictor to determine all the application configurations that will give the required utility, along with their resource demands for the next job (expressed in system-independent terms).
2. For each application configuration above, the coordinator sends the resource demand and allocation information to the CPU and network internal adaptors.
3. These adaptors independently perform an internal adaptation to determine their minimal energy configurations and the resultant energy for the given application configuration and resource allocation. (It is possible for these internal adaptations to use known information about the resource allocations to other applications as well.)

   For example, the CPU internal adaptor receives the CPU time allocated to the job and the predicted instruction count for the job. The adaptor calculates the ratio of these terms as the maximum TPI allowed. It then simply chooses the CPU configuration that has TPI less than the above maximum with the minimum energy-per-instruction (EPI). The total estimated energy is this EPI times the instruction count.
4. The internal adaptors convey their minimal energy configurations and resultant energy back to the per-application coordinator. It is possible that there is no feasible configuration for a given set of parameters, in which case, the internal adaptor returns a negative response.
5. The coordinator sums the total energy from the CPU and network, and chooses the application configuration (and associated hardware and network configurations) with the lowest total energy. It indicates the chosen configurations back to the internal adaptors of the application, hardware, and network respectively.

   If no feasible system configuration is found for any application configuration, the application coordinator indicates this to the global coordinator. At this point, the global coordinator may decide to drop the job, or trigger a global reallocation if too many jobs of this application have already been dropped, or the job may proceed anyway until its budget is exhausted and then move to best-effort mode.

[1] Some applications explicitly contain jobs of different types; e.g., I, P, and B frames of MPEG.

[2] The intuition is that the nature of the work done is generally the same for all jobs (quantified by TPI); only the amount of work varies across jobs (quantified by instruction count). For applications where this observation does not hold, the instruction count and TPI may not be independent, and other system-independent statistics (e.g., distribution of instructions) may be needed.

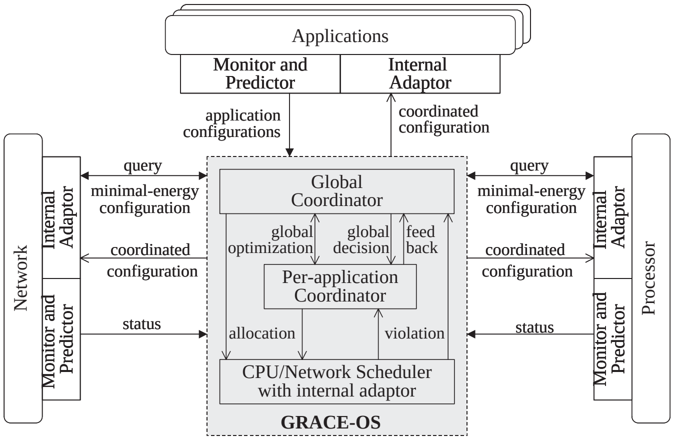


Figure 2: The GRACE system.

A key attribute of the optimization process described is that the coordinator is oblivious to the actual adaptation process within each layer, and the application and system layers are oblivious to each other. Thus, this process preserves the desirable isolation and independence of the different system layers and the applications. The only information exchanged between the different parts of the system is the resource demand of an application configuration (in system-independent terms) and the resource consumption of a system layer (in absolute terms). No part of the system needs to know about how any of the final configurations are reached, or even what they are.

Global optimization. Global adaptation needs to determine, for each application, configurations for the application and system such that system utility is maximized within the available resources. This is an NP-hard problem in general. Our previous work has solved this problem for some system scenarios by mapping it to the knapsack problem and using established heuristics to solve it [13, 17] (Section 3.1). The most significant limitation of our solutions so far has been that we assume that network bandwidth is unconstrained (although we charge for energy for network transmission [13, 14]).

Our future global optimizer will combine our hierarchical approach with search algorithms such as genetic algorithms or simulated annealing. It will search the space of possible resource allocations across applications and the utility demanded of each application. For each such combination, the optimizer will invoke the per-application optimizers independently for each application. These optimizers will return the best configurations using their optimization algorithms, if a solution exists. The global coordinator continues the search until no better feasible solution is likely to be returned.

In our implementations, the global optimizer took an order of magnitude more time than the per-application optimizer [13].

2.4 Putting it Together

We next summarize the operation of the complete GRACE framework, illustrated in Figure 2. In addition to the components already mentioned, the system includes monitors to measure application resource usage and utility, so the scheduler can ensure that all applications stay within their allocation and utility contract. The monitors are also used by the predictors to estimate system-independent and system-dependent portions of the resource demands. The global and per-application coordinators and the CPU and network schedulers with their internal adaptors are implemented as part of the operating system. To describe the full operation of GRACE, we start when a new application joins the system. This results in the following actions.

1. A global adaptation is triggered. This first adaptation process starts by initializing the system-independent and system-dependent parts of the resource demand predictions of the different configurations of the application (not shown in Figure 2). This part can be run in either best-effort mode, or a portion of the real-time partition of the system may be pre-allocated to the global coordinator.
2. The global coordinator then starts its optimization process by communicating with the per-application coordinators (Section 2.3). For each resource allocation suggested by the global coordinator, the per-application coordinators send back the best combination of configurations.
3. The coordinator chooses the per-application resource allocation and utility that maximizes system utility. It sends this information to each per-application coordinator, to the CPU and network schedulers, and the utility information to the application's predictor and internal adaptor. If no allocation returns a feasible solution, the new application is denied admission, and the system proceeds as before.
4. The scheduler schedules the next job, modifying its allocations based on prior underruns/overruns.
5. Before the job runs, the per-application coordinator determines the best CPU, network, and application configuration using the process described earlier (i.e., by determining the new allocation from the scheduler and then communicating with the internal adaptors). It conveys the best configurations to all the internal adaptors, which invoke the appropriate adaptations.
6. As the job runs, internal finer-granularity adaptations take place within the CPU, network, and application. The monitors monitor resource usage and utility. If a job violates its contract, the global coordinator is informed, and it decides if the job should be dropped or allowed to continue in best effort mode or if a new global adaptation should be triggered to reallocate resources. The monitors also help to update the predictions for the next job and for the next global adaptation.
7. If an application leaves the system or a resource reduces significantly (detected by monitors or by observing increased deadline misses), a global adaptation is triggered.

3 Results

We separately examine the benefits of global, per-application, and internal adaptations, using the prediction and optimization mechanisms discussed earlier. Our implementations so far use a simplistic network layer model and assume unconstrained network bandwidth, but we consider network transmission energy in Section 3.2. An implementation integrating all the adaptation modes and an adaptive network layer with constrained network bandwidth is currently in progress.

3.1 Benefits of Global Adaptation

We first evaluate the benefits of global adaptation of the GRACE framework based on its first prototype, GRACE-1. This prototype coordinates CPU frequency/voltage scaling in hardware, soft real-time CPU scheduling in OS, and quality adaptation in applications for stand-alone devices [17].

GRACE-1 takes an energy-greedy heuristic approach for global adaptation; specifically, the coordinator seeks to maximize the overall system utility under two constraints: (1) *CPU resource constraint*— the aggregate CPU utilization of all concurrent applications is ≤ 1. (2) *Energy constraint*— the battery should last for a user-defined lifetime (e.g., the time length of a DVD movie or expected runtime of all applications).

We have implemented GRACE-1 on a laptop with a single AMD Athlon processor, supporting six frequencies {300, 500, 600, 700, 800, 1000 MHz}. The OS is Redhat 7.2 Linux with a modified kernel 2.4.18; specifically, we add kernel modules for soft real-time scheduling and frequency setting.

We use adaptive MPEG video player applications, which can trade off video quality for CPU demand. Each player supports nine quality configurations by changing the frame rate and dithering methods (e.g., in color or gray). For each configuration, we profile the number of cycles for each frame decoding and use the average across all frames as the CPU demand of the configuration; we then define the utility as a linear function of the CPU demand; i.e., $u = a + b \times \frac{C}{P}$, where a and b are constants, and P and C are the period and demanded cycles of the configuration.

Since we currently do not have power meters, we use normalized energy from the AMD CPU specification, where the relative CPU power is 0.22, 0.35, 0.47, 0.6, 0.74, and 1.0 for the six different frequencies, respectively. When the CPU runs for t time units at frequency f, its normalized energy consumption is $t \times p(f)$, where $p(f)$ is the relative power at f.

We compare GRACE-1 with a baseline policy without adaptation, and with four policies that adapt one or two layers:

- *No-adapt*. No layer adapts: CPU runs at the highest frequency; the applications run at the highest configuration.
- *CPU-only*. Only the CPU is adaptive: CPU frequency is adjusted according to the total CPU demand of all concurrent applications.
- *App-only*. Only applications are adaptive: when an application arrives, it configures its configuration as high as possible, given the currently available CPU resource.
- *App-CPU*. This is uncoordinated adaptation in the application and hardware layer: the application and CPU adapt as in the *app-only* and *CPU-only* cases respectively.
- *App-OS*. Upon an application joining or leaving, the OS coordinates the adaptation of all current applications to maximize the system utility at the highest frequency.

Under each policy, we start an MPEG player every 12 seconds; each player decodes the video 4Dice.mpg with frame size 352×240 pixels and 1679 frames, and exits after completing the video. We measure the achieved battery lifetime and accumulated system utility for each policy. The accumulated system utility is the integral of the system utility over the achieved lifetime. We assume the desired lifetime is 900 seconds, and perform experiments with different battery energy of 300, 600, 900, and 1200 normalized units.

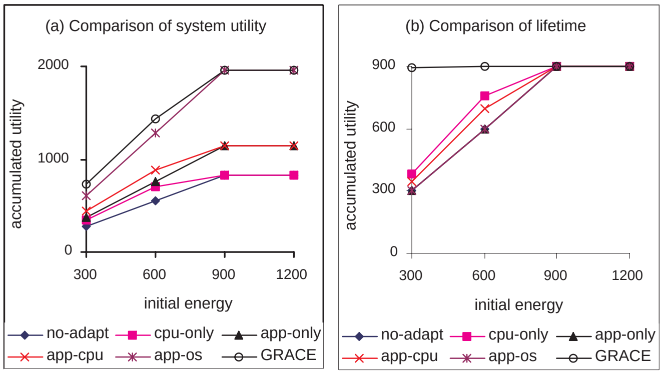


Figure 3: Benefits of global adaptation.

The results (Figure 3) show that GRACE-1 achieves the highest system utility and almost the desired lifetime for different energy availability. In particular, it improves the utility by 19%-63% and the lifetime by 33%-58%, compared to other policies that adapt only some of the system layers. This indicates that significant benefits are obtained from global adaptation in the GRACE framework. More detailed results can be found in [17].

3.2 Benefits of Per-Application Adaptation

We evaluated the benefits from per-application adaptation using adaptive applications and hardware. We consider CPU time and energy, and network transmission energy, but assume that network bandwidth is unconstrained. These results assume that a global coordinator (such as discussed above) has made a resource allocation for the adaptive application. A more detailed discussion of the system and the results can be found in [14].

The application studied is a video encoder based on the TMN (Test Model Near-Term) 1.7 encoder, which encodes standards-compliant H.263 streams. It was modified to integrate a fast motion search and to provide 16 different tradeoffs between computation and compression [14] through the selective elimination of low-value motion comparison and DCTs.

We predict instruction count and byte count for the next job (frame) using a set of linear predictors. For every transition between the previous frame's application configuration and the next frame's application configuration, the linear predictors map the previous frame's instruction and byte count to an estimate for the next frame.

The CPU studied is a modern out-of-order superscalar general-purpose processor with a conventional cache/memory hierarchy. It employs both dynamic voltage and frequency scaling (DVS) and architecture adaptation. It supports four architectural configurations varying in instruction window size

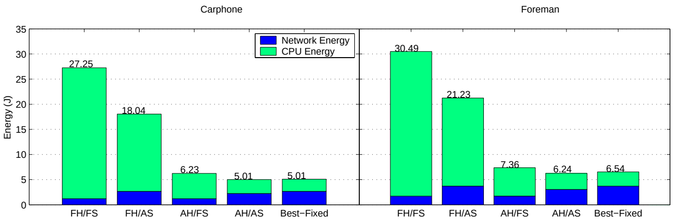


Figure 4: Energy comparison for fixed and adaptive systems

and number of functional units, and scales frequency between 100MHz and 1GHz at voltages ranging from 0.7 to 1.8 volts. The CPU's peak power dissipation in its fastest configuration is approximately 40W. Since we do not have access to CPUs that provide software control for architecture adaptation, we perform this study using a detailed cycle-by-cycle architecture simulator, RSIM.

The CPU profiles the application configurations for IPC and EPI as mentioned earlier and uses this information to convert instruction count to CPU time and energy.

To trade off between energy consumed in the network and energy consumed on the CPU, some model of the network is required. Our network model assumed a fixed cost (2×10^{-6} J) per byte, that any overhead is incurred on a per-frame basis and the bandwidth of the wireless network is much greater than the bandwidth consumed by the video stream. We also assumed that the network does not lose packets, and that energy consumed by the CPU running network code is insignificant compared to the per-byte energy cost.

We assume each job is assigned a CPU time (presumed to come from an external, global allocation) of 33ms.

Our simulations show that our approach, using per-application coordinated adaptation across layers, allows significant energy savings over systems that do not adapt all layers. The fine-grained adaptation offered by the hierarchical approach also allows additional energy savings by taking advantage of short-term variation in the stream.

Figure 4 shows the energy required to encode two sequences (*Carphone* and *Foreman*) under five hardware/application scenarios. In each chart, the first four bars show all the combinations of fixed and adaptive hardware and application respectively (e.g., FH/FA denotes fixed hardware running the application without any adaptation, FH/AA denotes fixed hardware and adaptive application). To see the benefits of the per-application adaptation over a best possible global application adaptation, we also tested a system where the adaptive hardware ran with a fixed application configuration where the thresholds are chosen to minimize energy across the specific encoded sequence, denoted "Best-Fixed."

In these experiments, the value of coordinated adaptation is clear; the addition of application adaptation saves significant amounts of energy. On the adaptive hardware, the addition of application adaptation saves an additional 20% or so more energy for *Carphone*, and around 15% more for *Foreman*. Comparing AH/AA with Best-Fixed, we also observe that the flexibility to choose different configurations on a frame-by-frame basis provides an additional energy savings of approximately 5% for our adaptive system when encoding *Foreman* (which consists of an initial "talking head" segment followed by a more complex "pan-and-zoom" segment). This additional savings from per-job adaptation is unachievable with a high-overhead, low-adaptation-frequency global adaptor alone. More short-term variations (e.g., use of I, P, B frames in MPEG or more variations in the input stream) will further amplify these benefits.

3.3 Benefits of CPU internal adaptation

To see the benefits of internal adaptation, we report results for internal adaptation in the CPU at the granularity of every 256 instructions. As in the previous section, the CPU can resize its instruction window and change the number of active functional units, but at a much finer granularity [15]. The CPU also employs DVS at a per-job granularity, but not within a job due to the relatively large overhead of invoking DVS. The study assumes a fixed time allocation for a job (assumed to come from the global coordinator), equal to the time it takes for the longest frame of the application on the fastest CPU configuration. It also assumes a fixed application (to isolate the benefits from internal CPU adaptation) and considers nine different multimedia applications encompassing speech, audio, and video codecs. The baseline is a system where the CPU adapts only at job boundaries (based on the predicted instruction count of the job and the globally allocated CPU time). We find that adding the above intra-job internal adaptation to the system with job-level adaptation gave an additional energy reduction ranging from 2% to 19% (average of 9%) across the nine applications. More details on this study can be found in [15] and another paper submitted to the same Computer special issue.[3]

4 Conclusions and Future Work

This paper describes the framework of the Illinois GRACE project, a novel approach for meeting the challenging demands of an increasingly dominant computing platform – *adaptive* mobile systems employing wireless communication and running multimedia applications. These systems have demanding, dynamic, and multidimensional resource constraints, along with strict limitations on available energy. The GRACE framework leverages the ability of multimedia applications to trade between output quality and resource consumption, and, to reduce energy by exploiting slack without affecting application quality.

We have proposed a system architecture where all layers (hardware and software) are flexible and adapt cooperatively to best meet the real-time demands of the applications, within the available resources. For the systems we target, we believe that such joint cross-layer adaptation will be critical to achieving future benefits from adaptation. In contrast to the limited previous work on joint adaptation, we believe that our framework will enable cross-layer cooperation in a manner that greatly enhances software reusability, enables a globally optimal solution, and readily enables adaptation across multiple layers. At

[3] The other submission is focused on internal hardware adaptation.

the same time, our approach retains the advantages of previous fixed systems that isolate functionality within different layers.

Furthermore, we have implemented significant portions of the complete framework outlined here. This work shows that the combination of global, per-application, and per-layer internal adaptations can provide significant reductions in the amount of energy consumed by multimedia systems. We have also shown that the proper use of coordination between operating system resource allocation and individual adaptive applications allows us to increase the total utility of the running system, while restricting energy consumption to achieve a specified battery lifetime. We are currently working on a complete system that integrates all levels of adaptation, incorporates adaptive networking with constrained bandwidth, uses more refined notions of utility, and considers applications with multiple synchronized tasks that share resource allocations.

Although our focus has been on energy management for single multimedia mobile nodes, we believe that the principles of the GRACE framework will be extensible to other domains as well, including non-adaptive and non-realtime applications, adaptations in other parts of the system (e.g., other hardware resources), cross-layer thermal management, and, through the addition of an additional cross-node adaptation layer, distributed applications across multiple nodes.

References

[1] S. Brandt and G. J. Nutt. Flexible soft real-time processing in middleware. *Real-Time Systems*, 22(1-2), 2002.

[2] M. Corner, B. Noble, and K. Wasserman. Fugue: time scales of adaptation in mobile video. In *Proc. of Multimedia Computing and Networking Conf.*, Jan. 2001.

[3] E. de Lara, D. Wallach, and W. Zwaenepoel. HATS: hierarchical adaptive transmission scheduling for multi-application adaptation. In *Proc. of Multimedia Computing and Networking Conf.*, Jan. 2002.

[4] C. Efstratiou et al. A platform supporting coordinated adaptation in mobile systems. In *Proc. of IEEE Workshop on Mobile Computing Sys. and Applications*, June 2002.

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

[6] 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.

[7] R. Kravets, K. Schwan, and K. L. Calvert. Power-Aware Communication for Mobile Computers. In *6th Intl. Workshop on Mobile Multimedia Communications*, 1999.

[8] T. Kunz et al. Image transcoding for wireless WWW access: The user perspective. In *Proc. of Multimedia Computing and Networking Conf.*, Jan. 2002.

[9] C. Lee et al. A scalable solution to the multi-resource QoS problem. In *Proc. of 20th IEEE Real-Time Systems Symp.*, 1999.

[10] B. Noble et al. Agile application-aware adaptation for mobility. In *Proc. of Symp. on Operating Systems Principles*, Dec. 1997.

[11] C. Poellabauer, H. Abbasi, and K. Schwan. Cooperative run-time management of adaptive applications and distributed resources. In *Proc. of 10th ACM Multimedia Conf.*, Dec. 2002.

[12] C. Rusu, R. Melhem, and D. Mosse. Maximizing the system value while satisfying time and energy constraints. In *Proc. of 23rd Real-Time Systems Symp.*, Dec. 2002.

[13] D. Sachs et al. Integrated global and local cross-layer adaptation for mobile multimedia systems. Tech. report, Univ. of Illinois. Submitted for publication, June 2003.

[14] D. G. Sachs, S. V. Adve, and D. L. Jones. Cross-layer adaptive video coding to reduce energy on general-purpose processors. In *Proc. of IEEE Intl. Conf. on Image Processing*, Sept. 2003.

[15] R. Sasanka, C. J. Hughes, and S. V. Adve. Joint Local and Global Hardware Adaptations for Energy. In *Proc. of the 10th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2002.

[16] M. Weiser et al. Scheduling for Reduced CPU Energy. In *Proc. of the 1st Symposium on Operating Systems Design and Implementation*, 1994.

[17] W. Yuan et al. Design and evaluation of cross-layer adaptation framework for mobile multimedia systems. In *Proc of Multimedia Computing and Networking*, Jan. 2003.

[18] H. Zeng et al. ECOSystem: Managing energy as a first class operating system resource. In *Proc. of 10th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.