GRACE-1: Cross-Layer Adaptation for Multimedia Quality and Battery Energy

Wanghong Yuan, *Member*, *IEEE*, Klara Nahrstedt, *Senior Member*, *IEEE*, Sarita V. Adve, *Member*, *IEEE*, Douglas L. Jones, *Fellow*, *IEEE*, and Robin H. Kravets, *Member*, *IEEE*

Abstract—Mobile devices primarily processing multimedia data need to support multimedia quality with limited battery energy. To address this challenging problem, researchers have introduced adaptation into multiple system layers, ranging from hardware to applications. Given these adaptive layers, a new challenge is how to coordinate them to fully exploit the adaptation benefits. This paper presents a novel cross-layer adaptation framework, called *GRACE-1*, that coordinates the adaptation of the CPU hardware, OS scheduling, and multimedia quality based on users' preferences. To balance the benefits and overhead of cross-layer adaptation, GRACE-1 takes a hierarchical approach: It *globally* adapts all three layers to large system changes, such as application entry or exit, and *internally* adapts individual layers to small changes in the processed multimedia data. We have implemented GRACE-1 on an HP laptop with the adaptive Athlon CPU, Linux-based OS, and video codecs. Our experimental results show that, compared to schemes that adapt only some layers or adapt only to large changes, GRACE-1 reduces the laptop's energy consumption up to 31.4 percent while providing better or the same video quality.

Index Terms—Energy-aware systems, support for adaptation, real-time systems, embedded systems.

1 Introduction

Battery-powered mobile devices that primarily process multimedia data, such as image, audio, and video, are expected to become important platforms for pervasive computing. For example, we can already use a smartphone to record and play video clips and use an iPAQ pocket PC to watch TV. Compared to conventional desktop and server systems, these mobile devices need to support multimedia quality of service (QoS) with limited battery energy. There is an inherent conflict in the design goals for high QoS and low energy: For high QoS, system resources such as the CPU often show high availability and utilization, typically consuming high power; for low QoS, resources would consume low power but yield low performance.

Although the requirement of high QoS and low energy is challenging, it is becoming achievable due to the strong advances in the *adaptable* system layers, ranging from hardware to applications. For example, mobile processors from Intel and AMD can run at multiple speeds, trading off performance for energy. Similarly, multimedia applications can gracefully adapt to resource changes while keeping the user's perceptual quality meaningful. Researchers have therefore introduced QoS and/or energy-aware *adaptation* into different system layers.[1] Hardware adaptation dynamically reconfigures system resources to save energy while providing the requested resource service and performance [3], [4], [5], [6], [7]. OS adaptation changes the policies of allocation and scheduling in response to application and resource variations [1], [8], [2], [9]. Finally, application adaptation changes multimedia operations or parameters to trade off output quality for resource usage or to balance the usage of different resources [10], [11], [12].

The above adaptation techniques have been shown to be effective for both QoS provisioning and energy saving. However, most of them adapt only a single layer or two joint layers (e.g., OS and applications [13], [14] or hardware [15], [16], [17]), as shown in Fig. 1a. More recently, some groups [2], [18], [19], [20] have proposed *cross-layer adaptation*, in which all layers adapt together in a coordinated manner, as illustrated in Fig. 1b. These cross-layer approaches, however, adapt only at coarse time granularity, e.g., when an application joins or leaves the system.

We believe that it is also necessary for a cross-layer adaptive system to adapt at fine time granularity, e.g., in response to small changes in the processed multimedia data. The Illinois GRACE project is developing a novel cross-layer adaptation framework that adapts multiple system layers at multiple time granularities. This paper presents the first generation implementation, called *GRACE-1*. GRACE-1 coordinates the adaptation of the CPU speed in the hardware layer, CPU scheduling in the OS layer, and multimedia quality in the application layer in response to system changes at both fine and coarse time granularity. The challenging problem addressed in GRACE-1 is as follows: *Given all adaptive layers, how do we coordinate*

- *W. Yuan is with DoCoMo USA Labs, 181 Metro Dr, Suite 300, San Jose, CA 95110. E-mail: whyuan@illinoisalumni.org.*
- *K. Nahrstedt, S.V. Adve, and R.H. Kravets are with the Department of Computer Science, University of Illinois, Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801. E-mail: {klara, sadve, rhk}@cs.uiuc.edu.*
- *D.L. Jones is with the Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801. E-mail: dl-jones@uiuc.edu.*

Manuscript received 31 Aug. 2004; revised 20 Jan. 2005; accepted 2 Mar. 2005; published online 16 May 2006.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0255-0804.

1. This paper focuses on three layers—hardware, OS, and applications—of stand-alone mobile devices such as portable video players. We also consider middleware systems such as Puppeteer [1] and Dynamo [2] as parts of the OS.

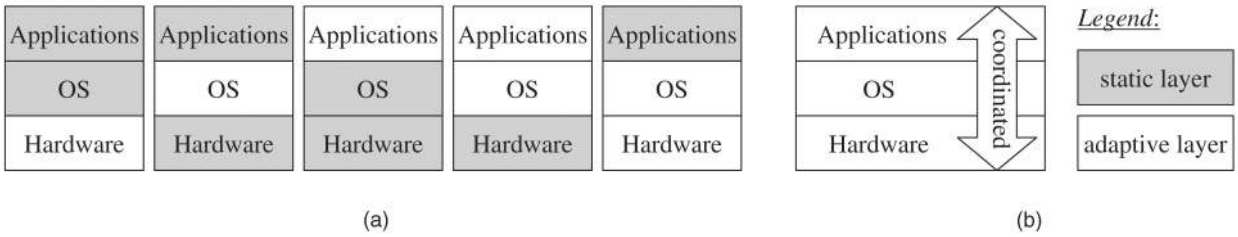

Fig. 1. Adaptation in different layers. (a) Previous work adapts one or two layers at a time, while (b) this paper considers coordinated cross-layer adaptation.

them to achieve the benefits of cross-layer adaptation with acceptable overhead.

To address this problem, GRACE-1 applies a *global* and an *internal* adaptation hierarchy, balancing the scope and the temporal granularity. Global adaptation coordinates all three layers in response to large system changes at coarse time granularity, e.g., when an application starts or exits. The goal of global adaptation is to achieve a systemwide optimization based on the user's preferences, such as maximizing multimedia quality while preserving the battery for a desired lifetime. On the other hand, internal adaptation adapts a single layer to small changes at fine granularity, e.g., when an MPEG decoder changes the frame type. The goal of internal adaptation is to provide the globally coordinated multimedia quality with minimum energy.

This paper makes three major contributions. First, we propose and justify a hierarchical framework for cross-layer adaptation. This framework consists of optimization-based coordination for all three layers and adaptive control for the CPU hardware and OS layers. Second, we design and implement a cross-layer adaptive system for stand-alone mobile devices. To the best of our knowledge, GRACE-1 is the first real system that integrates and coordinates adaptation in the CPU hardware, OS, and application layers. Finally, and more importantly, we perform a case study of a cross-layer adaptive system and analyze its impact on QoS and energy. In particular, we have validated GRACE-1 on an HP laptop with an adaptive Athlon CPU, Linux-based OS, and video codecs. The experimental results show that, compared to schemes that adapt only some layers or only at coarse and medium time scales, GRACE-1 reduces the total energy of the laptop by 1.4 percent to 31.4 percent, depending on application scenarios, while providing better or same video quality.

The rest of the paper is organized as follows. Section 2 introduces models of adaptive layers and system changes that trigger adaptation. Section 3 presents the design of GRACE-1, focusing on its architecture and adaptation hierarchy. Sections 4 and 5 show the implementation and experimental evaluation, respectively. Section 6 compares GRACE-1 with related work. Finally, Section 7 concludes this paper.

2 System Models

Our target systems are stand-alone mobile devices that primarily run CPU-intensive multimedia applications for a single user. This section introduces the adaptive models for the CPU hardware, OS allocation, and multimedia applications, and discusses what kinds of changes GRACE-1 should adapt to. Although GRACE-1 is currently built on these specific models, it can be extended to support other adaptive models for I/O and network. Such an extension is a part of our ongoing work.

2.1 Adaptive CPU Model

In the hardware layer, we consider reducing CPU energy. In general, CPU energy can be saved by switching the idle CPU into the lower-power sleep state or by lowering the speed (frequency and voltage) of the active CPU. The first approach, however, does not apply to our target multimedia applications, which access the CPU periodically (e.g., every 30 ms) and, hence, cause a short idle interval in each period. These idle intervals are often too short to put the CPU into sleep due to the switching overhead. This paper therefore focuses on the second approach, i.e., dynamic frequency/voltage scaling (DVS).

Specifically, we consider mobile processors, such as Intel's Pentium-M and AMD's Athlon, that can run at a discrete set of speeds, $\{f_1, \cdots, f_K\}$, trading off performance for energy. The CPU power (energy consumption rate) is dependent on the operating voltage [21]. When the speed decreases, the CPU can operate at a lower voltage, thus consuming less power. Since our goal is to reduce the total energy consumed by the whole device, rather than CPU energy only, we are more interested in the total power consumed by the device at different CPU speeds. Without loss of generality, we assume that the total device power decreases as the CPU speed decreases, i.e., the CPU power reduction is greater than the additional (if any) power consumed by other resources such as memory due to the CPU speed reduction. If this assumption does not hold, we will never choose the CPU speed that results in more total power but provides lower performance than another speed. In general, the relationship between the speed f and the total device power $p(f)$ can be obtained via measurements. Table 1, for example, shows the relationship, measured with an Agilent oscilloscope, for an HP N5470 laptop with a single Athlon CPU. During the measurements, an MPEG video player reads data from the local disk, decodes the data, and displays the decoded frame; the network is turned off.

2.2 Adaptive Application Model

We consider multimedia tasks (processes or threads) such as audio and video codecs that are long-lived (e.g., lasting

TABLE 1
Speed-Power Relationship for an HP N5470 Laptop

CPU speed f (MHz)	300	500	600	700	800	1000
Total power $p(f)$ (Watt)	22.25	25.84	28.24	31.05	35.44	39.06

TABLE 2
CPU Demand for Different Dithering Methods

dithering method	gray	mono	color
CPU cycles	1.28×10^7	1.64×10^7	1.82×10^7

for minutes or hours) and CPU-intensive (i.e., the time spent for I/O access is negligible relative to the CPU time). Each task consumes CPU resource and generates an output. *Adaptive* tasks can trade off output quality for CPU demand by changing multimedia operations or parameters [22], [12], [14]. For example, `mpegplay`, an adaptive MPEG decoder, can decode the video with different dithering methods. Different dithering methods need different numbers of CPU cycles for the same frame (Table 2). We refer to the set of quality levels a task supports as its *quality space*, Q, which may be continuous or discrete.

For any quality level $q \in Q$, a task releases a job (e.g., decoding a frame) every period $P(q)$. The period is the minimum time interval between successive jobs and can be specified by the task based on its semantics such as the rate to read, process, and write multimedia data. Each job has a soft deadline, typically defined as the end of the period. By *soft* deadline, we mean that a job should, but does not have to, finish by this time. In other words, a job may miss its deadline. Multimedia tasks are soft real-time tasks and, hence, need to meet some percentage of job deadlines. This percentage can be specified by the application developers or users based on application characteristics (e.g., audio needs to meet more deadlines than video).

2.3 Adaptive Allocation Model

A task consumes CPU cycles when executing each job. To meet the deadline, the task needs to be allocated some CPU cycles for each job. However, different jobs of the same task may demand a different amount of cycles due to variations in the input data (e.g., I, P, and B frames). Unlike hard real-time tasks, *soft* real-time multimedia tasks do not need worst-case-based allocation. We therefore periodically allocate to each task a statistical number of cycles, $C(q)$, which can be estimated with our previously developed kernel-based profiler [23]. For example, if a video decoder requires meeting 95 percent of deadlines and, for a particular video stream and quality level, 95 percent of frame decoding demands no more than 9×10^6 cycles, then the parameter $C(q)$ is 9×10^6. Correspondingly, the allocated processing time to the task is $\frac{C(q)}{f}$ per period if the CPU runs at the speed f.

Combining the adaptive CPU, OS, and application models together, we get a cross-layer adaptation model (Fig. 2). Specifically, we need to configure the CPU speed in the hardware layer, the CPU allocation to each task in the OS layer, and the quality of each task in the application layer.

2.4 Adaptation Triggers

Mobile systems often exhibit dynamic variations, which trigger adaptation in the GRACE-1 system. This paper considers two kinds of variations, changes of the number of tasks (i.e., task entry or exit) and changes in the input data of a task. These two kinds of variations occur at different time scales and have different impact: The former requires reallocating CPU among tasks at coarse granularity (e.g., in minutes or per-task), while the latter changes CPU usage a little bit at fine granularity (e.g., in tens of milliseconds, per-job, or cross multiple jobs for a scene change). An adaptive system needs to respond to the small changes in the latter case; otherwise, these small changes may cause deadline miss, thus degrading multimedia quality, or idle the CPU, thus wasting energy.

3 Hierarchical Cross-Layer Adaptation

This section presents the design of the GRACE-1 cross-layer adaptation framework. We describe the architecture of GRACE-1 and its major operations.

3.1 Overview

GRACE-1 is a cross-layer adaptation framework that coordinates the adaptation of the CPU speed, OS scheduling, and multimedia quality for stand-alone mobile devices. Fig. 3 shows the architecture of GRACE-1, which consists of five major components: a coordinator, a task scheduler, a CPU adapter, a battery monitor, and a set of task-specific adaptors. The coordinator coordinates all three layers based on the available energy, task quality levels, and user's preferences. Each task has a specific task adapter, which adjusts the operations or parameters for the task. The CPU adapter minimizes the CPU speed and total power while providing the required performance. The scheduler enforces the coordinated allocation for all tasks. It also monitors the cycle usage of each task and adapts the CPU allocation in response to the usage changes.

The key problem addressed in GRACE-1 is as follows: *Given the three adaptive layers, how do we coordinate them to*

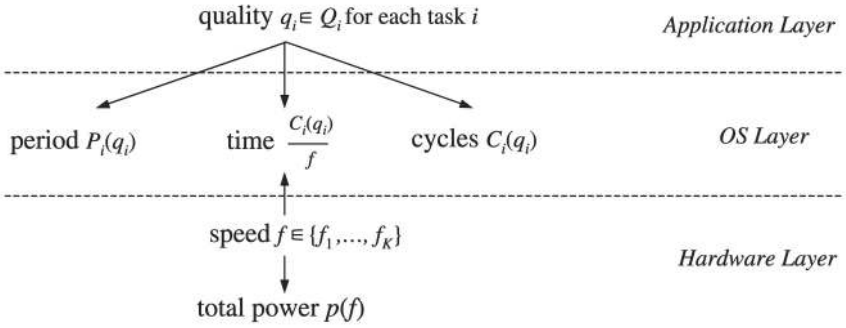

Fig. 2. Cross-layer adaptation for adaptive CPU speed, CPU allocation, and multimedia quality.

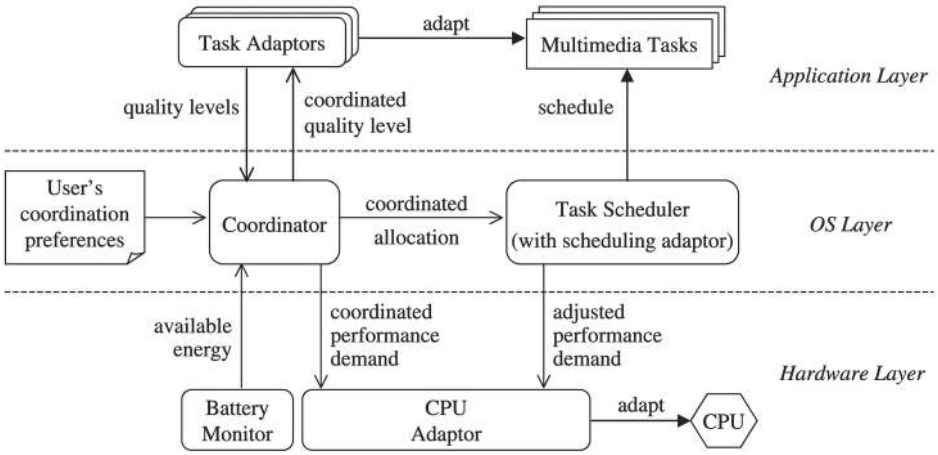

Fig. 3. Architecture of the GRACE-1 cross-layer adaptation framework.

achieve the benefits of the cross-layer adaptation with acceptable overhead? GRACE-1 takes three steps to address this problem. First, when a task joins or leaves the system, GRACE-1 uses *global adaptation* to decide the quality level and CPU allocation for each task and the average power consumption of the device. These global decisions try to achieve a systemwide optimization, such as maximizing the overall multimedia quality for a desired battery lifetime. Second, GRACE-1 uses *speed-aware real-time scheduling* to enforce the globally coordinated decisions. Finally, GRACE-1 uses *internal adaptation* to adapt the CPU allocation and/or the CPU speed in response to the changes in the CPU usage of individual tasks. The goal of the internal adaptation is to minimize energy consumption while enabling each task to operate at the coordinated quality level. We next discuss the three major operations in detail.

3.2 Global Adaptation

Global adaptation happens when the CPU resource needs to be reallocated among tasks, e.g., due to the entry or exit of a task. In such a case, GRACE-1 coordinates all three layers (the CPU hardware, OS allocation, and multimedia quality) to achieve a system-wide optimization based on the preferences of the user of the device. The user may have different preferences for trading off multimedia quality against energy in a battery-powered device. For example, the user may want to maximize multimedia quality when the battery is high and minimize power consumption to achieve a desired lifetime (e.g., finishing a two-hour movie) when the battery is low.

The coordinator takes the user's preferences as an input for the global adaptation (Fig. 3). Although GRACE-1 can support different user preferences, this paper uses a representative preference, called *lifetime-aware max-quality min-power*, that considers battery lifetime, multimedia quality, and power consumption together. In this preference, the user wants 1) to maintain the battery for a desired lifetime, 2) to maximize the current multimedia quality, and 3) to minimize the total power consumed by the device. The desired lifetime can be specified by the user based on, e.g., how long tasks should run before recharging the battery. If the user does not specify the desired lifetime, the coordinator uses a very short lifetime to relax the lifetime constraint.

More formally, let us assume that 1) there are n adaptive tasks running concurrently, 2) each task $i, 1 \leq i \leq n$, demands $C_i(q_i)$ cycles per period $P_i(q_i)$ for a quality level q_i in its quality space Q_i, and 3) the residual battery energy is E, the desired lifetime is T, and the total power of the device is $p(f)$ at the CPU speed f. The global coordination problem for the *lifetime-aware max-quality min-power* preference is to select a quality level q_i for each task and a CPU speed f such that

$$\begin{aligned}
&\text{maximize} && QF(q_1, ..., q_n) && \text{(overall quality function)}, && (1)\\
&\text{minimize} && p(f) && \text{(total device power)}, && (2)\\
&\text{subject to} && p(f) \times T \leq E && \text{(lifetime constraint)}, && (3)\\
& && \sum_{i=1}^{n} \frac{\frac{C_i(q_i)}{f}}{P_i(q_i)} \leq 1 && \text{(CPU constraint)}, && (4)\\
& && q_i \in Q_i && i = 1, ..., n, && (5)\\
& && f \in \{f_1, ..., f_K\}, && && (6)
\end{aligned}$$

where (4) is the CPU scheduling constraint. This constraint requires that the total CPU utilization of all tasks is no more than 1. The reason is that GRACE-1 uses an earliest-deadline-first (EDF)-based scheduling algorithm, which will be discussed in Section 3.3.

Equation (1) denotes the overall quality of all concurrent tasks. Now, the problem is how to quantify the overall quality. Although there is a lot of related work (such as utility functions [24], [14]) on measuring multimedia quality from the user's point of view, it is still challenging to quantify the perceptual quality of an adaptive multimedia task and the overall quality of concurrent tasks. Instead of quantifying the perceptual quality, GRACE-1 characterizes multimedia quality in a qualitative way through a *weighted max-min* allocation approach, which is commonly used in network bandwidth allocation [25]. Intuitively, a task delivers higher quality with more CPU allocation. The overall quality of concurrent tasks can be reduced to the level of the most important task; e.g., the movie quality is bad with great video but poor audio.

Specifically, we make two assumptions: First, at the same CPU speed f, each adaptive task increases its output quality as its CPU utilization

$$\frac{\frac{C_i(q)}{f}}{P_i(q)}$$

There are n concurrent tasks. Each task $i, 1 \le i \le n$, has a quality space Q_i, a weight w_i, a minimum CPU utilization $U_i^{min}(f)$ and maximum utilization $U_i^{max}(f)$ at CPU speed f.

1. Initialization
   - 1.1 Set the allowable CPU speed $f = \max\{f : p(f) \times T \le E \text{ and } f \in \{f_1, ..., f_K\}\}$.
   - 1.2 Allocate each task i its minimum CPU utilization $U_i = U_i^{min}(f)$ at the allowable speed f. If the minimum allocation is impossible, global adaptation fails and rejects the new task.
   - 1.3 Define the unsatisfied task set as $\Gamma = \{i : 1 \le i \le n \text{ and } U_i < U_i^{max}(f)\}$.
2. Repeat until $\Gamma = \phi$ or the total CPU utilization $\sum_{i=1}^{n} U_i = 1$
   - 2.1 For each task $i \in \Gamma$, increase its allocation U_i by $\frac{w_i}{\sum_{i \in \Gamma} w_i}(1 - \sum_{i=1}^{n} U_i)$. If the task is satisfied with $U_i \ge U_i^{max}(f)$, set $U_i = U_i^{max}(f)$ and remove the task from Γ.
3. Global adaptation results
   - 3.1 Application layer: Configure each task to the highest quality allowed by its CPU allocation, i.e., $q_i = \max\{q \in Q_i \text{ and } \frac{\frac{C_i(q)}{f}}{P_i(q)} \le U_i\}$.
   - 3.2 OS layer: Allocate each task $C_i(q_i)$ cycles per period $P_i(q_i)$. Note that if a task has a discrete set of quality levels, its actual utilization $\frac{\frac{C_i(q)}{f}}{P_i(q)}$ may be less than U_i.
   - 3.2 Hardware layer: Set the CPU to the lowest speed that meets the total CPU demand, i.e., $f_{coord} = \min\left\{f : \sum_{i=1}^{n} \frac{\frac{C_i(q)}{f}}{P_i(q)} \le 1 \text{ and } f \in \{f_1, ..., f_K\}\right\} \le f$.

Fig. 4. Algorithm for global adaptation.

increases. This assumption is reasonable; otherwise, the task will never run at the quality level that demands more CPU but provides lower quality than another level. Second, at each speed f, each adaptive task has a minimum CPU utilization $U_i^{min}(f)$ and a maximum utilization $U_i^{max}(f)$ for the lowest and highest quality level, respectively. With these assumptions, GRACE-1 makes global adaptation as follows:

- The coordinator finds the allowable speed,

  $$f = \max\{f : p(f) \times T \le E \text{ and } f \in \{f_1, ..., f_K\}\}$$

  that allows the battery to last for the desired lifetime T.
- The coordinator initially allocates to the tasks their minimum CPU utilization at the allowable speed, $U_i^{min}(f)$ and increases their CPU allocation proportional to their weight (importance to the user) until all tasks have the maximum utilization, $U_i^{max}(f)$, or their total CPU utilization becomes 100 percent. This weighted max-min allocation policy makes sense for mobile devices since they often have a single user who can prioritize concurrent tasks.
- Based on this coordinated CPU allocation, each task then configures its QoS parameters, such as frame rate.[2] If a task supports only a discrete set of quality levels, the task is configured to the highest quality level allowed by the CPU allocation.

Fig. 4 shows the global coordination algorithm. This algorithm provides an approximate solution. Its complexity is $O(n^2 + \sum_{i=1}^{n} m_i + K)$, where n is the number of concurrent tasks, m_i is the number of discrete quality levels of task i ($m_i = 1$ if task i can change quality continuously), and K is the number of CPU speeds.

2. Although not explicit here, Grace can support quality consistency of dependent tasks (e.g, lip synchronization among audio and video) by treating these tasks as a task group and adapting each group jointly [26].

3.3 Speed-Aware Real-Time Scheduling

After global adaptation, GRACE-1 needs to enforce the global decisions on multimedia quality and power consumption. In particular, each task should provide the coordinated quality and the device should consume no more than the coordinated power. To enforce these decisions, GRACE-1 uses a variable-speed constant bandwidth server (VS-CBS) scheduling algorithm [27]. This algorithm is extended from the CBS algorithm [28] for an energy-aware context.

Specifically, when a task joins, the OS creates a VS-CBS for the task. The VS-CBS is characterized by four parameters: a period P, a maximum cycle budget C, a budget c, and a deadline d. The maximum budget and period equal the allocated number of cycles and period, respectively, of the served task. The budget is initialized as the maximum budget, and the deadline is initialized as the deadline of the first job. The scheduler always selects a VS-CBS with the earliest deadline. As the selected VS-CBS executes a job, its budget c is decreased by the number of cycles the job consumes. That is, if the VS-CBS executes for Δt time units at speed f, its budget is decreased by $\Delta t \times f$. Whenever c is decreased to 0, the budget is recharged to C and the deadline is updated as $d = d + P$. At that time, the VS-CBS may be preempted by another one with an earlier deadline.

Note that the deadline of the VS-CBS may be different from the deadline of the current job executed by the server. Compared to approaches that use job deadline and allocate cycles to the job directly, VS-CBS is better to handle overruns (i.e., a job needs more cycles than allocated). In particular, these approaches typically protect overruns by running the overrun job in best effort mode until the next period begins [29]. The VS-CBS algorithm, instead, postpones the VS-CBS deadline. If the VS-CBS still has the earliest deadline, it continues to execute the job, which increases the possibility that the overrun job meets its deadline.

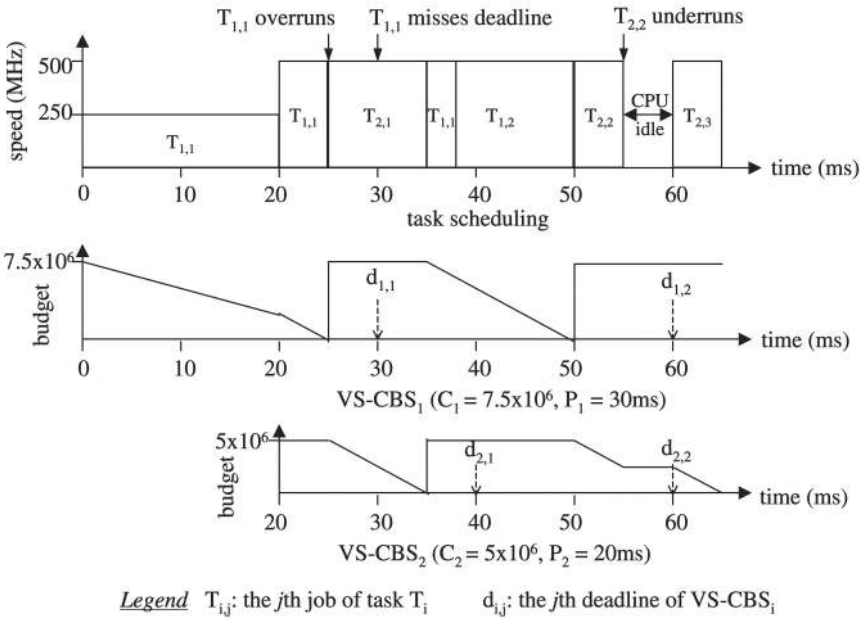

Fig. 5. An example of the VS-CBS scheduling algorithms.

Fig. 5 shows an example of the VS-CBS scheduling algorithm. Initially, when task T_1 joins at time 0, the coordinator performs a global adaptation. As a result of the global adaptation, T_1 is allocated 7.5×10^6 cycles per period 30 ms and the CPU speed is set to $\frac{7.5\times10^6}{30} = 250$ MHz. VS-CBS$_1$ is created for T_1, initializes its budget to 7.5×10^6 cycles, and its deadline to $d_{1,1} = 30$ ms. VS-CBS$_1$ executes T_1's first job, $T_{1,1}$, which has deadline 30 ms. At time 20 ms, task T_2 joins and another global adaptation happens. As a result, the CPU speed is increased to 500 MHz and VS-CBS$_2$ is created for T_2. VS-CBS$_2$ initially has budget 5×10^6 and deadline $d_{2,1} = 40$ ms. Since VS-CBS$_1$ has the earliest deadline, it continues to execute until time 25, when its budget becomes 0. The budget of VS-CBS$_1$ is then recharged to 7.5×10^6 cycles and its deadline is updated as $d_{1,2} = d_{1,1} + 30 = 60$ ms. At this time, VS-CBS$_2$ has the earliest deadline, so it starts to execute job $T_{2,1}$.

This example illustrates that the VS-CBS algorithm enforces the coordinated allocation at the coordinated speed and provides isolation among tasks. This algorithm, however, cannot efficiently handle overruns and underruns (i.e., a task needs fewer cycles than the allocated). An overrun may result in deadline miss and, hence, degrade quality. An underrun, on the other hand, may idle the CPU and, hence, waste energy, for example, when job $T_{1,1}$ misses its deadline due to overrun. Similarly, when job $T_{2,2}$ underruns at time 55, the CPU becomes idle. We next discuss how to use internal adaptation to handle overruns and underruns.

3.4 Internal Adaptation

GRACE-1 performs internal adaptation to handle small variations in the CPU usage of each task. In general, internal adaptation can happen in each of the hardware, OS, and application layers. For example, multimedia tasks can internally adapt QoS parameters within an acceptable range of the globally coordinated quality level, e.g., through rate control [1], [12]. The CPU hardware can also adapt internally to save more energy since the coordinated speed may be larger than the total CPU demand due to the discrete speed options. For example, Ishihara and Yasuura [30] proposed a simulation approach that provides the required performance by executing each cycle (or a group of cycles) at two different speeds.

This paper does not discuss the internal adaptation in the application and hardware layers for two reasons: 1) The internal application adaptation is often application-specific and 2) when used in real implementations, the above internal CPU adaptation may incur large overhead since the cycle division results in frequent speed changes and interrupts. Instead, we focus on the internal OS adaptation and its consequent CPU adaptation. The basic idea of the internal OS adaptation is to adjust the CPU allocation (and possibly the CPU speed) of each task based on its runtime CPU usage. In particular, we investigate two approaches, *per-job* adaptation and *multijob* adaptation. The former adjusts the cycle budget for the current job of a task upon an overrun or underrun, while the latter adjusts the cycle budget for all later jobs of a task in case of consistent overruns or underruns.

3.4.1 Per-Job Adaptation

In per-job adaptation, the scheduler allocates an extra budget to or reclaims the residual budget from a job when it needs more or less cycles than allocated. Specifically, let us consider a task T_i underrun at time t with a residual budget of b_i cycles. This residual budget would be wasted since the task has no job to execute until the start of the next period, t'. To avoid this waste, the scheduler reclaims the residual budget from the VS-CBS. This reclamation enables a lower CPU speed. At the current speed f, which can be the globally coordinated speed or the speed adapted in previous internal adaptations, the original total cycle demand in the time interval $[t, t']$ is $f \times (t' - t)$, but the new total cycle demand becomes $f \times (t' - t) - b_i$ due to the

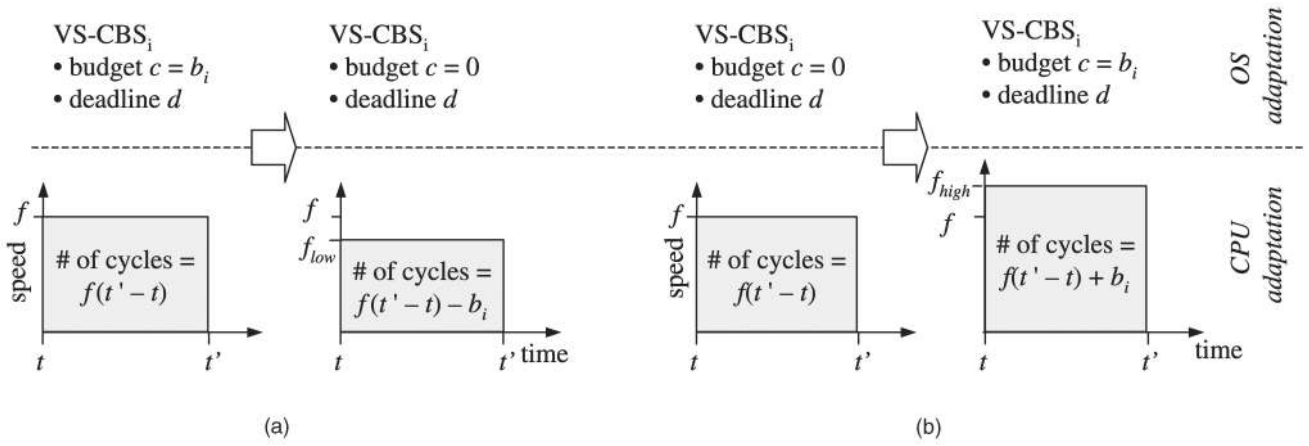

Fig. 6. Per-job adaptation for handling underruns and overruns. (a) Reclaim budget to handle underrun. (b) Allocate extra budget to handle overrun.

reclamation. As a result, the speed can be decreased to $f_{low} = f - \frac{b_i}{t'-t}$ in the interval $[t, t']$.

Similarly, consider a task T_i overrun at time t. To enable the task to finish the overrun job by its deadline t', we can allocate an extra budget to the serving VS-CBS, rather than recharging its budget and postponing its deadline. The number of extra cycles, however, is known only after the job finishes. We, therefore, heuristically predict that the current job needs the same amount of extra cycles as the last overrun job of the task. For a predicted overrun of b_i cycles, the total budget demand over the interval $[t, t')$ becomes $f \times (t' - t) + b_i$. To support this extra allocation, the CPU needs to run at a higher speed $f_{high} = f + \frac{b_i}{t'-t}$.

Fig. 6 illustrates the per-job adaptation. The idea of underrun handing is similar to previous reclamation approaches [31], [16], [32]. The idea of accelerating the CPU to handle overrun is new. The per-job adaptation shows the flexibility of our speed-aware real-time scheduling algorithm: The scheduler can handle underruns and overruns by changing the CPU speed without affecting the CPU allocation of other tasks.

3.4.2 Multijob Adaptation

In the global adaptation, the coordinator makes decisions on CPU allocation according to the statistical cycle demand of each task. This statistical demand may change over time due to variations in the input data (e.g., scene changes). Fig. 7, for example, plots the variations of the instantaneous and statistical cycle demands of an MPEG decoder when it plays video `4dice.mpg` with frame size 352×240 pixels. The decoder's statistical cycle demand, defined as the 95th percentile of the job cycles, changes for different video segments. For example, the 95th percentile of all jobs is much higher than that of the first and last 300 jobs but is lower than that of the middle 300 jobs.

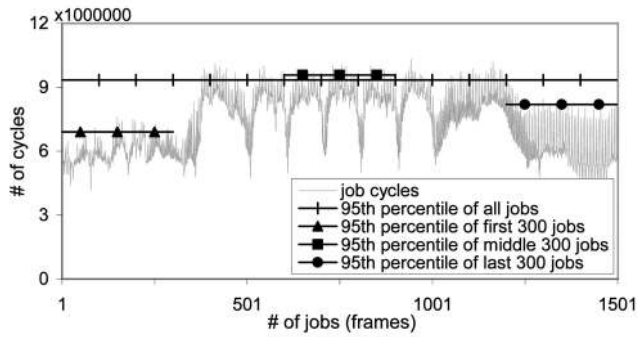

Fig. 7. Variations of the instantaneous and statistical cycle demand of an MPEG video decoder.

The dynamic nature of the statistical demand implies that a multimedia task may consistently underrun or overrun its coordinated allocation. The consistent underruns or overruns would trigger the above per-job adaptation frequently. Such frequent adaptation is inefficient due to the cost associated with each speed change (see Section 5.2). To avoid this, GRACE-1 triggers *multijob* adaptation to update the statistical demand of the task (and, hence, the allocation to all later jobs of the task) according to its recent CPU usage.

Specifically, the scheduler uses a profiling window to keep track of the number of cycles each task has consumed for its recent W jobs, where W is the window size (W is set to 100 in our implementation). When the overrun or underrun ratio of a task exceeds a threshold, the scheduler calculates a new statistical cycle demand, e.g., as the 95th percentile of the job cycles in the profiling window. Let C' be the new statistical cycle demand. The scheduler then uses an exponential average strategy, commonly used in control systems [10], [33], to update the task's statistical demand C as $\alpha \times C + (1 - \alpha) \times C'$, where $\alpha \in [0, 1]$ is a tunable parameter and represents the relative weight between the old and new cycle demands (α is set to 0.2 in our implementation). Consequently, the scheduler will update the maximum cycle budget of the serving VS-CBS.

When the multijob adaptation updates a task's statistical cycle demand, the total CPU demand of all concurrent tasks changes accordingly. If the total demand exceeds the allowable CPU speed, the multijob adaptation fails. After reaching a certain failure threshold, the scheduler can either tell the task to degrade its quality and CPU requirements or trigger a global adaptation to reallocate the CPU among all tasks. GRACE-1 takes the latter approach since it can potentially achieve a better configuration. For example, if an important task, such as a user-focused video, consistently overruns and the CPU already runs at the maximum speed,

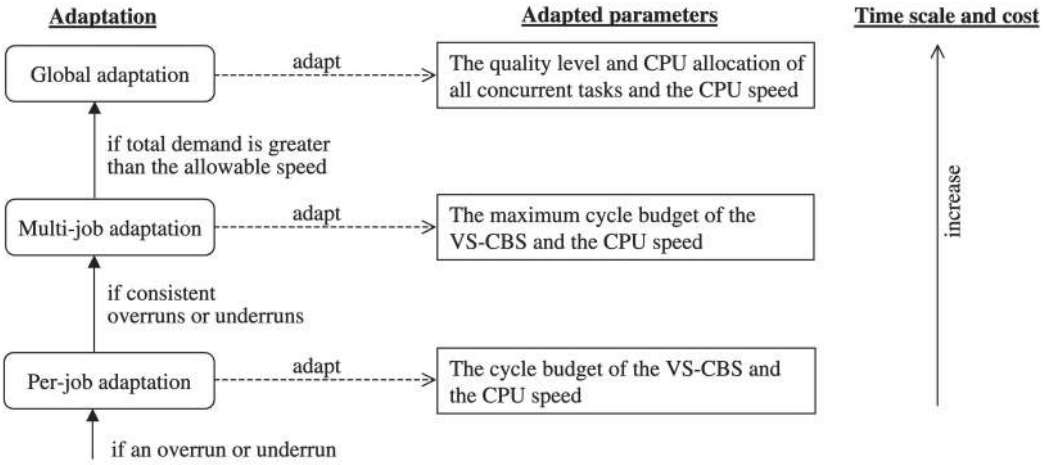

Fig. 8. Applying various adaptations at different time scales to handle CPU usage variations.

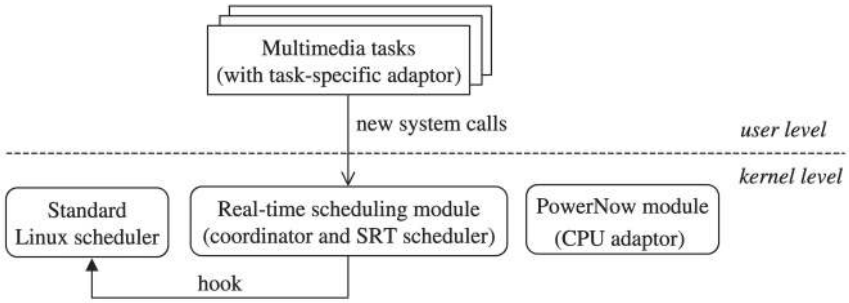

Fig. 9. Software architecture of GRACE-1 implementation.

TABLE 3
New System Calls for GRACE-1

System Call	Description	Sample Code
`enterSRT`	Enter soft real-time mode and demand CPU resource guarantees.	`enterSRT()` *// QoS setup* `setQoS()` *// execute jobs periodically* while (! complete) { `getQoS()` reconfigure QoS if changed `doAJob()` `finishJob()` } `exitSRT()`
`setQoS`	Set CPU demand for a quality level. A global adaptation is invoked after the calling task finishes QoS setup.	
`getQoS`	Get coordinated QoS level, based on which the calling task configures its QoS parameters.	
`finishJob`	Tell the kernel the calling task has finished a job and wait until the next period.	
`exitSRT`	Exit soft real-time mode and release CPU.	

GRACE-1 can allocate more cycles to the important task by decreasing the allocation to other less important tasks.

In summary, to handle variations in the CPU usage of individual tasks, GRACE-1 integrates three different adaptations: per-job adaptation, multijob adaptation, and global adaptation, and applies them at different time scales. Fig. 8 illustrates this integration.

4 Implementation

We have implemented a prototype of the GRACE-1 cross-layer adaptation framework. The hardware platform is an HP Pavilion N5470 laptop with a single AMD Athlon CPU [34]. This CPU supports six different speeds: 300, 500, 600, 700, 800, and 1,000 MHz, and its speed and voltage can be adjusted dynamically under operating system control. The operating system is Red Hat 8.0 with a modified version of Linux kernel 2.6.5, as discussed below.

Fig. 9 illustrates the software architecture of the prototype implementation, which is similar to the design architecture in Fig. 3. The entire implementation contains 2,605 lines of C code, including about 185 lines of modification to the Linux kernel. The task adapter is application-specific and, hence, is integrated into the application task. In the Linux kernel, we add two loadable modules, one for the CPU adapter and one for the coordinator and soft real-time (SRT) scheduler. The `PowerNow` module (the CPU adapter) changes the CPU speed by writing the speed and corresponding voltage to a system register `FidVidCtl` [34].

The real-time scheduling module (the coordinator and SRT scheduler) is hooked into the standard Linux

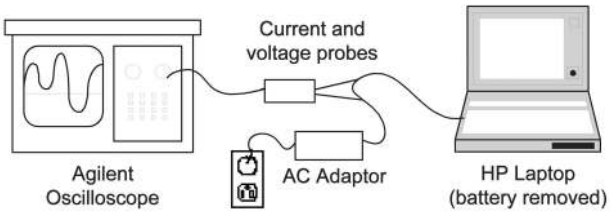

Fig. 10. Power measurement with a digital oscilloscope.

scheduler, rather than replacing the latter. In doing so, we can support the coexistence of real-time and best-effort applications and also minimize the modification to the kernel. Table 3 lists new system calls for multimedia tasks to communicate with the kernel and illustrates how to use these system calls. These new system calls are designed for adaptive applications. They, however, become a limitation for legacy applications that cannot be modified. To support legacy applications, GRACE-1 can be enhanced as follows: First, the OS kernel can derive application requirements and control application behavior. For example, the scheduler can derive the period of applications by monitoring their CPU usage pattern [35] and further control their execution rate via CPU allocation. Second, middleware such as Puppeteer [1] can be used to adapt applications without open source.

The SRT scheduler is time-driven. To improve the granularity of soft real-time scheduling, we add a *high resolution timer* [36] with resolution $500\ \mu s$ into the kernel and attach the SRT scheduler as the call-back function of the timer. As a result, the SRT scheduler is invoked every $500\ \mu s$. When the SRT scheduler is invoked, it charges the cycle budget of the current task's VS-CBS, updates its deadline if necessary, and sets the scheduling priority of the current task based on its VS-CBS's deadline. After that, the SRT scheduler invokes the standard Linux scheduler, which in turn dispatches the task with the highest priority.

5 Experimental Evaluation

This section experimentally evaluates the GRACE-1 cross-layer adaptation framework. We describe the experimental setup and then present the overhead of GRACE-1 and its benefits of global and internal adaptation. These results demonstrate that GRACE-1 achieves the benefits of the cross-layer adaptation with acceptable overhead.

5.1 Experimental Setup

Our experiments are performed on an HP N5470 laptop with 256 MB RAM and without network connection. We use an Agilent 54621A oscilloscope to measure the energy consumed by the laptop. Specifically, we remove the battery from the laptop and measure the current and voltage of the AC power adaptor, as shown in Fig. 10. The total power consumed by the laptop is the product of the measured current and voltage and the energy consumption is the integral of the power over time.

The experimental applications include an H263 video encoder and an MPEG video decoder, both of which are single-threaded. The H263 encoder, based on the TMN (Test Model Near-Term) tools, supports three quality levels with different quantization parameters: 5, 18, and 31. All three levels encode a frame every 150 ms. Before encoding each frame, the H263 encoder retrieves the coordinated quality level from the kernel and sets the quantization parameter correspondingly.

The MPEG decoder, based on the Berkeley MPEG tools, supports four quality levels with different dithering methods: `gray`, `mono`, `color`, and `color2`. All four levels decode a frame every 50 ms. When adapting the dithering method, the MPEG decoder restarts to decode the video from the current frame number with the new dithering method. This quality adaptation may incur a large overhead due to the restart. The MPEG decoder also uses the X library to display the decoded image. To address the dependency between the MPEG decoder and the X server, we let them share a VS-CBS, which executes the decoder most of the time but executes the X server when it is called by the decoder. Correspondingly, the SRT scheduler uses the priority inheritance protocol [37] to set the scheduling priority of the X server to that of the decoder when the decoder calls the X library.

For each quality level of the above two codecs, we use our previously developed kernel-based profiler [23] to profile the number of cycles for each frame processing and estimate the statistical cycle demand as the 95th percentile cross all frames. This demand enables each codec to meet about 95 percent of deadlines. The input for the MPEG decoder is `StarWars.mpg` with frame size 320×240 pixels and 3,260 frames. The input for the H263 encoder is `Paris.cif` with 1,065 frames. Table 4 summarizes the quality levels of these two codecs. When these codecs start, they tell the above parameters to the kernel. The kernel then stores them into the process control block of the corresponding codec.

5.2 Overhead

In the first set of experiments, we analyze the overhead of GRACE-1. Specifically, we measure the time cost for global adaptation, real-time scheduling, internal adaptation, and new system calls. Unless specified otherwise, we run the CPU at the lowest speed, 300 MHz, to measure the time elapsed during each operation. This elapsed time represents

TABLE 4
Quality Levels for Two Adaptive Multimedia Codecs

	MPEG decoder				H263 encoder		
	dithering				quantization level		
quality level	gray	mono	color	color2	31	18	5
period (ms)	50	50	50	50	150	150	150
cycles ($\times 10^6$)	12.77	16.02	17.83	20.07	55.06	61.76	90.18
bandwidth (MHz)	255.4	320.4	356.6	401.4	367.1	411.7	601.2

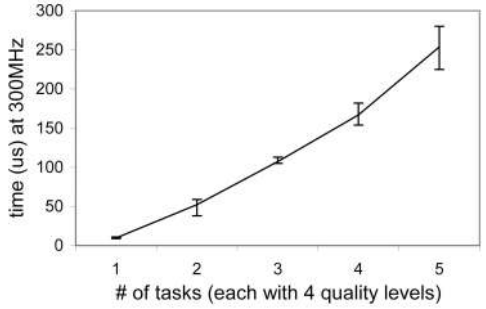

Fig. 11. Cost of global adaptation. The solid line shows the mean of six measurements and the error bars show the minimum and maximum of the six measurements.

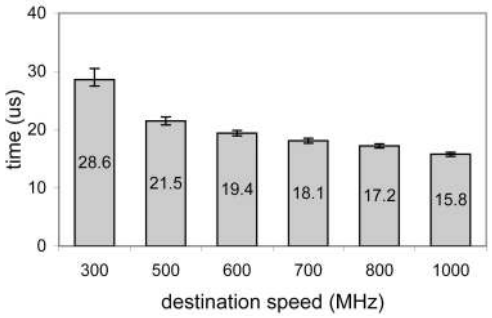

Fig. 13. Cost of changing CPU speed. The solid line shows the mean of 12 measurements and the error bars show the minimum and maximum of the 12 measurements.

the worst-case cost in terms of different CPU speeds. Although we are unable to directly measure the energy cost (i.e., energy consumed during each operation), our results imply that it is small since the time cost is small.

To measure the cost for global adaptation, we run one to five MPEG decoders at a time (mobile devices seldom run more than five active applications concurrently) and measure the time elapsed for coordinating the CPU, OS, application layers and determining their configuration (Steps 1 and 2 in Fig. 4). The results in Fig. 11 show that the cost for global adaptation increases significantly with the number of tasks, but is quite small (below $300\ \mu$s) for up to five tasks. GRACE-1, however, cannot invoke global adaptation frequently for two reasons. First, the cost reported here does not include time for configuring each layer based on the global decisions. This time may be large, especially in the application layer, e.g., the MPEG decoder takes hundreds of milliseconds to change its dithering method. Second, frequent global adaptation may result in rapid fluctuation of the perceived quality, which could be annoying to the user.

To measure the cost for soft real-time scheduling, we run one to five MPEG decoders at a time and measure the time elapsed for each invocation of the SRT scheduler. Fig. 12 plots the results. The scheduling cost is below $4\ \mu$s and, hence, negligible for multimedia processing. In terms of relative overhead, the scheduling cost is below 0.8 percent since the granularity of soft real-time scheduling is $500\ \mu$s (recall that we use a $500\ \mu$s-resolution timer to invoke the SRT scheduler). Further, the cost of real-time scheduling does not increase significantly with the number of concurrent tasks. The reason is that, like the $O(1)$ scheduling algorithm in Linux kernel 2.6, our real-time scheduler also uses an $O(1)$ algorithm, which primarily maintains the status of the current task.

Now, we analyze the cost of the adaptation in the CPU and operating system layers. To measure the cost for CPU speed change, we adjust the CPU from one speed to another and measure the time elapsed for each adjustment, during which the CPU cannot perform computation. Fig. 13 plots the results. The cost for speed adaptation depends on the destination speed and is below $40\ \mu$s. This cost is acceptable for GRACE-1 to adapt the CPU speed at most twice per job, one for handling overrun or underrun and the other for recovering the speed at a new period.

To measure the cost for internal operating system adaptation, we run one MPEG decoder and measure the time elapsed during each per-job and multijob adaptation. The results (Fig. 14) show that multijob adaptation has a much larger overhead (factor of 100) than per-job adaptation. However, both per-job and multijob adaptations incur negligible overhead relative to multimedia processing. For example, the cost of multijob adaptation is below $22\mu s$, which is less than 0.05 percent of the time for decoding an MPEG frame.

Finally, we measure the cost for the system calls (Table 3). To do this, we run three MPEG decoders and measure the time elapsed during each system call in the application level. Fig. 15 plots the cost, which is negligible relative to multimedia processing for the following reasons: First, although `getQoS` is called once per job, the cost per call is very small. Second, although `enterSRT`, `setQoS`, and `exitSRT` have larger costs per call, they are called only once or a few times per task. Finally, although `finishJob`

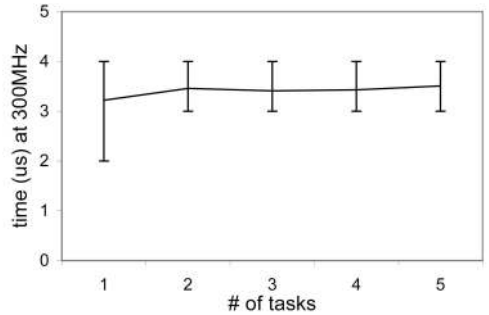

Fig. 12. Cost of soft real-time scheduling. The solid line shows the mean of 5,000 measurements and the error bars show the 95 percent confidence intervals.

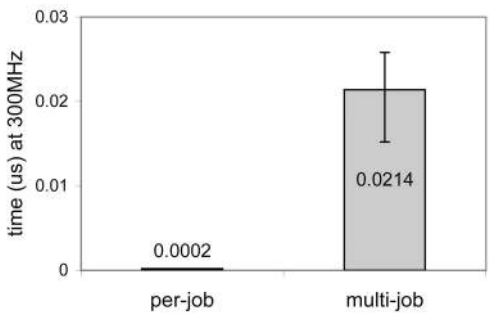

Fig. 14. Cost of internal adaptation. The bars show the mean of six measurements and the error bars show the minimum and maximum of the six measurements.

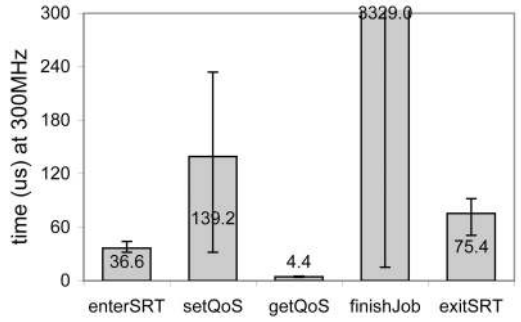

Fig. 15. Cost of new system calls. The bars show the mean of 10 measurements and the error bars show the minimum and maximum of the 10 measurements.

has a large cost (in milliseconds) per call, this cost does not matter from the QoS point of view for the following reason: Immediately after calling `finishJob`, the application is suspended in the kernel until the next period when the next job is available and `finishJob` returns from the kernel. In other words, the cost of `finishJob` includes the time when the application waits for the next period.

Another interesting result from Fig. 15 is that `setQoS` and `finishJob` both exhibit large deviations in their cost. For `setQoS`, it may trigger a global adaptation if the task sets the last quality level. For `finishJob`, the calling task is usually suspended until the next period, but starts a new job immediately if the task finishes the previous job at or after the deadline.

5.3 Benefits of Global Adaptation

We now analyze the benefits of GRACE-1's global adaptation for QoS provisioning and energy saving. To do this, we compare GRACE-1 with other adaptation schemes that adapt only some of the three system layers:

- *No-adapt*. This is a baseline system. The CPU runs at the highest speed. Each task operates at the highest quality level. The operating system does not handle overruns and underruns.
- *CPU-only*. Same as *no-adapt* except that the CPU adapts when a task joins or leaves. The CPU sets the speed to meet the total demand of all concurrent tasks, all of which operate at the highest quality level.
- *CPU-app*. Joint adaptation in the hardware and application layers. Each task adapts when it joins: It configures its quality level as high as possible given the available CPU resource when the task joins. The CPU adapts when a task joins or leaves: The CPU sets the speed to meet the total demand of all concurrent tasks.

For a fair comparison, GRACE-1 does not perform internal adaptation here. We perform two kinds of experiments under each adaptation scheme: 1) *single run*, in which we run each of the MPEG decoder and H263 encoder one at a time, and 2) *concurrent run*, in which we start an H263 encoder and start an MPEG decoder 60 seconds later. Table 5 shows the desired lifetime for the single and concurrent runs (i.e., the time until each experiment finishes). Although the experiment time is short, it is enough to evaluate GRACE-1. In the concurrent run, the H263 encoder and MPEG decoder have weights 0.8 and 1.0, respectively; a codec exits immediately if there is insufficient CPU resource. This concurrent run represents several realistic scenarios, such as a video-conferencing client that compresses the video captured at its own side and displays the video from the other clients, and a video recorder that plays back the recorded video while capturing new video.

In each experiment, we measure three metrics: *energy*, *achieved lifetime*, and *CPU allocation*. The last metric indicates multimedia QoS in a qualitative way based on the weighted max-min policy in which the overall quality is better if the minimum allocation to tasks is high. We do not measure the actual battery lifetime due to the difficulties in recharging the same battery energy for different adaptation schemes. We instead assign an energy budget and decrease it by the energy consumed by the laptop as the experiment runs. The achieved lifetime is the time interval until the energy budget is exhausted or no more task will run. We repeat the experiments with different energy budgets in terms of the percentage of the highest demanded energy that is sufficient for the CPU to always run at the highest speed for the desired lifetime.

Fig. 16 reports the achieved lifetime and energy consumption when the energy budget varies from 60 percent (which enables the CPU to run at the lowest speed for the desired lifetime) to 100 percent. In the single runs, GRACE-1 always achieves the desired lifetime and extends the lifetime by 6.4 percent to 38.2 percent when the energy budget is low. The reason is that GRACE-1 considers the energy constraint and is aware of the lifetime when coordinating the CPU, OS, and application layers in the global adaptation. In contrast, other schemes are oblivious to lifetime.

In terms of energy, GRACE-1 always consumes the lowest energy. Specifically, for the single H263 case with energy budget of 70 percent, GRACE-1 allocates CPU bandwidth 411 MHz to the H263 encoder for the desired lifetime, while other schemes allocate the highest CPU demand to the H263 but with shorter lifetime. We also notice that *CPU-only* and *CPU-app* extend the lifetime and save energy compared to *no-adapt*. This shows the benefits of CPU adaptation since the CPU does not need to always run at the highest speed.

In the concurrent run, GRACE-1 achieves the desired lifetime when the energy budget is no less than 80 percent, which is sufficient to concurrently run the H263 encoder and MPEG decoder at their lowest quality levels. In particular, when the energy budget is 80 percent, GRACE-1 extends the lifetime by 9.8 percent relative to *CPU-app*, which also runs two codecs together by adapting their quality but does not coordinate their adaptation. When the energy budget is 60 percent or 70 percent, the global

TABLE 5
Desired Lifetime for Single and Concurrent Runs

Experiment	H263 encoder	MPEG decoder	concurrent run
Desired lifetime (second)	160	163	223

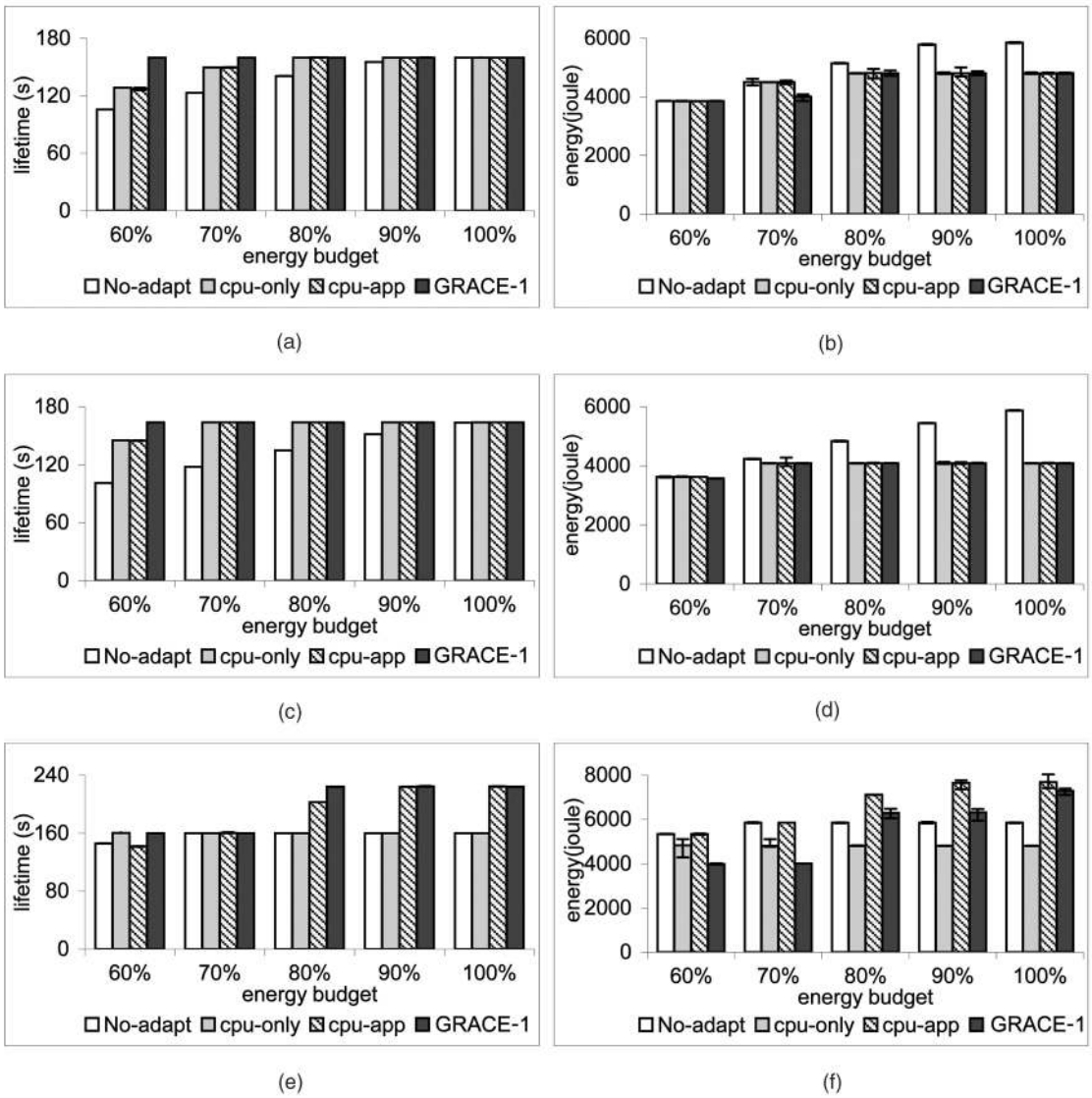

Fig. 16. Comparing GRACE-1 with systems adapting only some layers: The bars show the mean of five measurements and the error bars show the minimum and maximum of the five measurements. (a) Lifetime for H263 encoder. (b) Energy for H263 encoder. (c) Lifetime for MPEG decoder. (d) Energy for MPEG decoder. (e) Lifetime for concurrent run. (f) Energy for concurrent run.

adaptation succeeds when the H263 encoder starts, but fails when the MPEG decoder starts. This failure causes the MPEG decoder to be rejected. That is, GRACE-1 runs only the H263 encoder, thus resulting in a shorter lifetime. This shows that GRACE-1 is limited by few quality levels of codecs, i.e., these two codecs cannot run concurrently when the allowable speed is low due to the low energy budget.

In terms of energy, GRACE-1 reduces energy by 5.1 percent to 31.4 percent relative to *CPU-app*, though they run the same number of tasks. GRACE-1 consumes more energy than *no-adapt* and *CPU-only* when the energy budget is greater than 70 percent. The reason is that GRACE-1 runs two tasks while the latter two schemes run the H263 encoder task only with shorter lifetime.

After analyzing lifetime and energy, we next analyze next the CPU allocation to tasks. In the single runs, GRACE-1 limits the CPU speed for the desired lifetime and then allocates CPU to the single task based on the allowable speed, while other schemes always allocate the highest CPU demand to the single task and, hence, may use up the energy before the lifetime. We, hence, focus on the concurrent run. Figs. 17 and 18 show the concurrent allocation with energy budget of 80 percent and 100 percent, respectively.

Clearly, *no-adapt* and *CPU-only* are oblivious to application adaptation and allocate CPU only to the H263 encoder, which starts first. This is not desirable for concurrent execution. GRACE-1 and *CPU-app* both adapt applications to run two codecs concurrently. However, GRACE-1 coordinates the adaptation and allocates CPU in the user-specified weighted max-min fair manner. In particular, with an energy budget of 80 percent, GRACE-1 limits the total allocation (and, hence, CPU speed) for the desired lifetime and finishes two codecs; *CPU-app*, on the other hand, runs each codec at as high a quality as possible, but does not finish the MPEG decoder. When the energy budget is 100 percent and enables the highest CPU speed, GRACE-1 coordinates the allocation to the two codecs and has a higher minimum allocation than *CPU-app* in the time interval [60, 223] when two codecs run concurrently. This

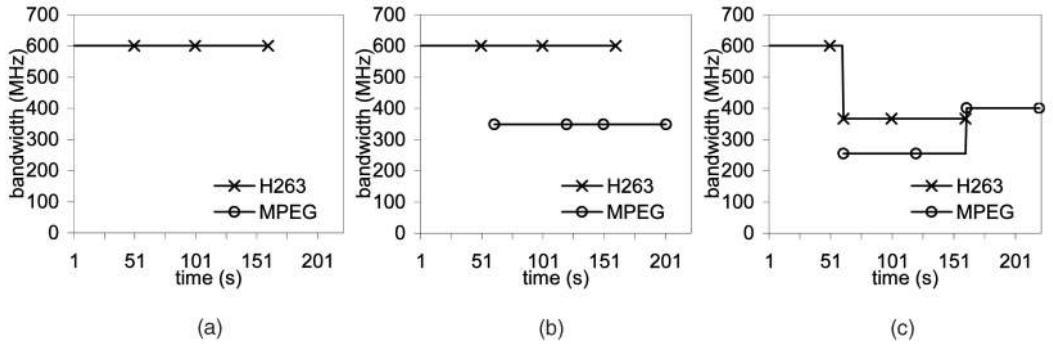

Fig. 17. CPU bandwidth allocation for concurrent run with energy budget of 80 percent. GRACE-1 coordinates allocation to achieve the desired lifetime. (a) No-adapt and CPU-only. (b) CPU-app. (c) GRACE-1.

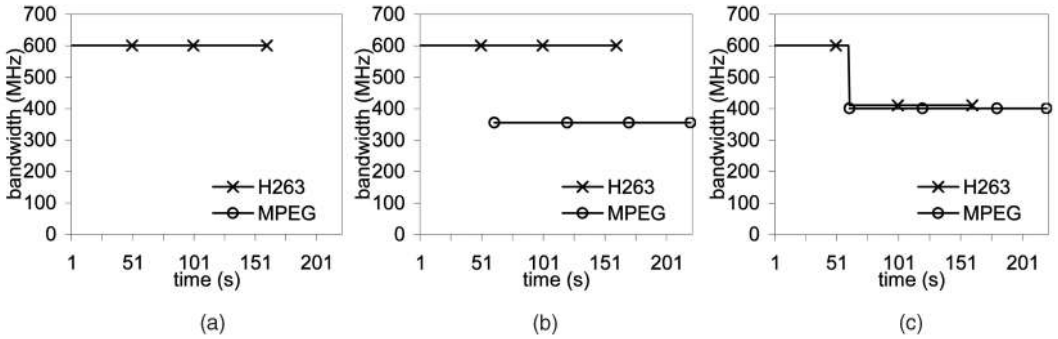

Fig. 18. CPU bandwidth allocation for concurrent run with energy budget of 100 percent. GRACE-1 coordinates allocation to increase the minimum allocation while achieving the desired lifetime. (a) No-adapt and CPU-only. (b) CPU-app. (c) GRACE-1.

implies that GRACE-1 achieves a better overall quality in terms of the weighted max-min policy.

5.4 Benefits of Internal Adaptation

We now analyze the benefits of GRACE-1's internal adaptation at fine time granularity. To do this, we compare GRACE-1 with the following schemes that perform global cross-layer adaptation only at coarse and medium time granularity:

- *Coarse-only*. It coordinates the adaptation of the CPU, OS, and applications when a task joins or leaves. This represents cross-layer adaptive systems (e.g., [18], [19], [20]) that handle only large system changes at coarse time scales.
- *Coarse-medium*. It is the same as *coarse-only* except that it also dynamically updates the CPU demand of each task based on its 95th percentile CPU usage of its recent 100 jobs and adjusts the CPU speed to reach a full utilization.

Note that, in the above two schemes and GRACE-1, applications do not perform internal adaptation (they adapt only when they join or leave), as discussed in Section 3.4. We repeat the above single and concurrent run experiments under the above two schemes and GRACE-1. Since all three schemes perform the same global adaptation, we focus on

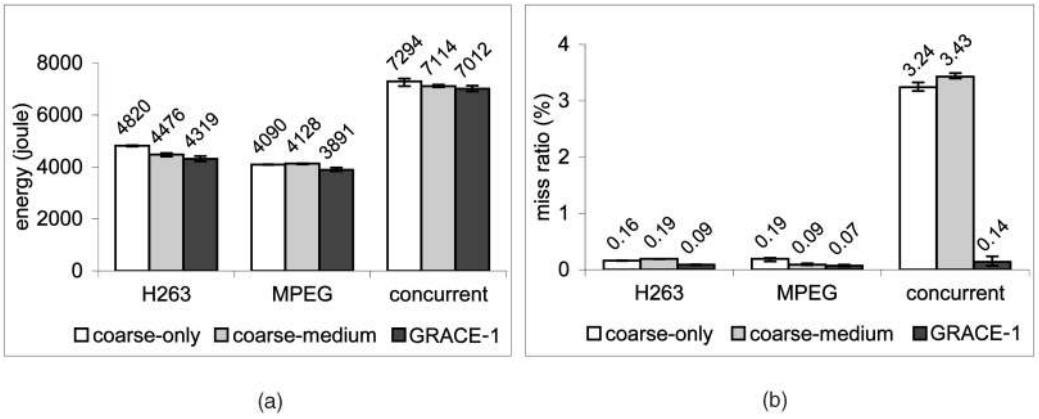

Fig. 19. Comparing GRACE-1 with systems adapting at coarse and medium time scales when energy budget is 100 percent: the bars show the mean of five measurements and the error bars show the minimum and maximum of the five measurements. (a) Energy consumption. (b) Deadline miss ratio.

the cases with energy budgets of 100 percent and measure energy consumption and deadline miss ratio. Fig. 19 reports the results. We notice immediately that GRACE-1 consumes the lowest energy and misses fewer deadlines. GRACE-1 saves energy by 3.8 percent to 10.4 percent relative to *coarse-only* and by 1.4 percent to 5.7 percent relative to *coarse-medium*. These energy benefits result from GRACE-1's internal adaptation for handling underruns. The underrun handling is effective since the budget reclamation decreases the total CPU demand and may, hence, allow the CPU to run at the next lower speed.

The expected deadline miss ratio would be about 5 percent since each codec is allocated CPU based on its 95th percentile of demand. All three schemes have a very low deadline miss ratio in the single runs, but use different approaches. *Coarse-only* and *coarse-medium* schemes utilize the unallocated cycles, which exist due to the discrete speed options, to implicitly handle an overrun. GRACE-1, on the other hand, explicitly controls an overrun by allocating an extra cycle budget. This handling is especially effective for the concurrent run in which GRACE-1 lowers the average deadline miss ratio of the two codecs by a factor of 22.8 when compared to *coarse-only* and *coarse-medium* schemes. The reason is that the two codecs may overrun at the same time and compete for the unallocated cycles without overrun handling.

5.5 Results Summary and Discussion

Overall, our experimental results show that GRACE-1 provides significant benefits for QoS provisioning and energy saving. Compared to adaptation schemes that adapt only some of the three layers, GRACE-1's global adaptation allocates CPU in a weighted max-min fair way for better quality, extends the lifetime by 6.4 percent to 38.2 percent when the battery is low, and saves energy by up to 31.4 percent when the battery is high. Compared to adaptation schemes that adapt all three layers only at coarse or medium time granularity, GRACE-1's internal adaptation further saves energy by 1.4 percent to 10.4 percent while missing fewer deadlines, especially for concurrent execution.

Although GRACE-1 does not measure perceptual quality from the user's point of view, it can accept the user's preferences (*lifetime-aware max-quality min-power* in the current implementation) during the coordination. GRACE-1 could extend the cross-layer adaptation with a user layer to support the changes of the user's preferences such as different utility functions.

We also found that the effectiveness of GRACE-1 is limited by few quality levels of our experimental applications. In particular, when the energy budget is low, GRACE-1 allows a lower CPU speed for the desired lifetime. The allowable speed is too low to support two concurrent codecs. We expect that GRACE-1 will provide more benefits if applications can adapt quality in a wider range.

6 Related Work

In this section, we first review QoS- and energy-aware adaptation approaches in various system layers. These approaches are leveraged by the GRACE-1 cross-layer adaptation framework. We then compare GRACE-1 with other frameworks that also coordinate adaptations.

6.1 QoS and/or Energy-Aware Adaptation

There have been numerous research contributions on adaptation in the hardware and software layers of mobile devices. Here, we summarize the work related to our GRACE-1 system. In the hardware, dynamic voltage scaling (DVS) [31], [38], [39], [32] is commonly used to save CPU energy by adjusting the frequency and voltage based on application workload. In general, the workload is heuristically predicted for best-effort applications [38], [7] or derived from the worst-case demands of hard real-time applications [39], [16]. These two approaches, however, cannot be directly applied to soft real-time multimedia applications, since the worst-case-based derivation is often too conservative for multimedia applications and the heuristic prediction may violate multimedia timing constraints too often. Grunwald et al. [38], for example, concluded that no heuristic algorithm they examined saves energy without degrading multimedia quality. In contrast to the above DVS work, GRACE-1 integrates DVS with real-time scheduling and, hence, saves energy while delivering soft deadline guarantees. This integration is similar to other work on OS-directed hardware adaptation [31], [15], [16], [17].

In the application layer, multimedia applications can gracefully adapt output quality against CPU and energy usage. Corner et al. [40] proposed three time scales of adaptation for video applications. Flinn et al. [1], [41] explored how to adapt applications that have open or closed source code to save energy. Similarly, Mesarina and Turner [11] discussed how to reduce energy in MPEG decoding. The above application adaptation work is orthogonal and complementary to GRACE-1. GRACE-1 further provides a mechanism to coordinate application adaptation with hardware and OS adaptation.

In the OS layer, much work has been done on real-time CPU resource management. Like GRACE-1, these resource managers, such as SMART [29], deliver soft deadline guarantees. Some of them also adapt to the variations in the CPU usage. Unlike GRACE-1, however, these approaches assume a static CPU speed without considering energy. Some groups have also researched OS or middleware services to support application adaptation. For example, Odyssey [12] adds system support for mobile applications to trade off data fidelity and energy. Agilos [10], DQM [22], PARM [2], and Puppeteer [1] are middleware systems that help applications adapt to resource variations. GRACE-1 provides similar support but differs from the above work in that GRACE-1 coordinates the adaptation of the CPU hardware, OS scheduling, and multimedia quality.

Recently, energy has become important in resource management. For example, ECOSystem [9], [42] and Nemesis [43] manage energy as a first-class OS resource. Vertigo [35] saves energy by monitoring application CPU usage and adapting the CPU speed correspondingly. Muse [44] saves energy for Internet hosting clusters by shutting down unnecessary servers. Real-time CPU scheduling has also been extended for energy saving. For example, Lee et al. [45] investigated how to reduce leakage power in fixed and dynamic priority scheduling algorithms. This approach is further integrated with DVS to minimize both static and dynamic energy [46]. More recently, some researchers have proposed energy-aware scheduling algorithms for dependent tasks [47], [48]. Jejurikar and Gupta

[47] proposed algorithms to compute the static slow-down factor of DVS for tasks sharing resources. Zhu et al. [48] proposed power-aware scheduling algorithm for tasks modeled with AND/OR graphs. The above related work is complementary to GRACE-1. For example, our previous work [26] shows that GRACE-1 can support the adaptation of dependent tasks with quality and execution dependency.

6.2 Adaptation Coordination

Other related work includes QoS and/or energy-aware resource allocation. Q-RAM [14] models QoS management as a constraint optimization that maximizes QoS while guaranteeing minimum resources to each application. Perez et al. [49] proposed a similar QoS-based resource management scheme. Park et al. [18] extended Q-RAM to optimize energy for multiresource, multitask embedded systems. Similarly, IRS [50] coordinates allocation and scheduling of multiple resources to admit as many applications as possible. Rusu et al. [20] proposed two optimization algorithms that consider constraints of energy, deadline and utility. These coordination approaches are similar to GRACE-1's global adaptation in that all of them coordinate the resource allocation to multiple applications for a systemwide optimization. None of the above work performs internal adaptation at fine time granularity.

Recently, some groups have also been researching adaptation coordination. Efstratiou et al. [13] proposed a middleware platform that coordinates multiple adaptive applications for a system-wide objective. Q-fabric [51] supports the combination of application adaptation and distributed resource management via a set of kernel-level abstractions. HATS [52] adds control over bandwidth scheduling to the Puppeteer middleware [1] and coordinates adaptation of multiple applications to improve network performance. The above related work considers application adaptation only (with the support of resource management in the OS or middleware). In contrast, GRACE-1 considers cross-layer adaptation of the CPU speed, OS scheduling, and application QoS.

More recently, there is some work on QoS and energy aware cross-layer adaptation [53], [54], [19], [17]. Pereira et al. [54] proposed a power-aware application programming interface that exchanges the information on energy and performance among the hardware, OS, and applications. This work is complementary to GRACE-1, e.g., GRACE-1 can be extended to manage I/O resources with this interface. PADS [17] is a framework for managing energy and QoS for distributed systems and focuses on the hardware and OS layers. Mohapatra et al. [53] proposed an approach that uses a middleware to coordinate the adaptation of hardware and applications at coarse time granularity (e.g., at the time of admission control). EQoS [19] is an energy-aware QoS adaptation framework. Like GRACE-1, EQoS also formulates energy-aware QoS adaptation as a constrained optimization problem. GRACE-1 differs from EQoS for two reasons: First, EQoS targets hard real-time systems where the application set is typically static and requires worst-case guarantees. In contrast, GRACE-1 aims for multimedia-enabled mobile devices. The soft real-time nature of multimedia applications offers more opportunities for QoS and energy trade-off, e.g., more energy can be saved via stochastic (as opposed to worst-case) QoS guarantees. Second, EQoS focuses only on global adaptation at coarse time granularity, while GRACE-1 uses both global and internal adaptation to handle changes at different time granularity. The global and internal adaptation hierarchy enables GRACE-1 to balance the benefits and cost of cross-layer adaptation.

7 Conclusion

This paper presents *GRACE-1*, a cross-layer adaptation framework to trade off multimedia quality against energy for stand-alone mobile devices that primarily run CPU-intensive multimedia applications. The challenging problem addressed in GRACE-1 is as follows: Given the adaptive CPU hardware, OS scheduling and multimedia applications, how do we coordinate them based on the user's preferences such as maximizing multimedia quality for a desired battery lifetime? To address this problem, GRACE-1 uses a novel hierarchy of *global* and *internal* adaptation. Global adaptation coordinates all layers at coarse time granularity when a task joins or leaves, while internal adaptation adapts the hardware and OS layers at fine granularity when a task changes CPU demand at runtime.

We have validated GRACE-1 on an HP N5470 laptop with an adaptive Athlon CPU, Linux OS, and MPEG and H263 video codecs. Our implementation has shown that cross-layer adaptation preserves the isolation of different layers; in particular, multimedia applications only need to add five new system calls to support the cross-layer adaptation. Our experimental results indicate that GRACE-1 achieves significant adaptation benefits with acceptable overhead. Specifically, for our implemented *lifetime-aware max-quality min-power* adaptation policy, GRACE-1 almost achieves the user-desired lifetime, reduces the total energy up to 31.4 percent, allocates CPU in a max-min fair way for better quality, and misses fewer deadlines when compared to adaptation schemes that adapt only some layers or only at coarse and medium time scales.

Our work with GRACE-1 taught us some lessons. First, we found that the efficiency of GRACE-1 was limited by few quality levels of our experimental applications. We expect that GRACE-1 will provide more benefits if applications can adapt quality in a wider range. Second, we found that the energy efficiency of GRACE-1 was limited by the few speed options of the Athlon CPU (i.e., the CPU may run at a higher speed than the total CPU demand, thus wasting energy, due to the discrete speed options). To address this limitation, we plan to emulate the optimal speed with two available speeds [30]. Finally, we are extending GRACE-1 to develop more coordination policies, manage other resources such as network bandwidth, and integrate internal adaptations of CPU architecture, network protocols, and applications.

Acknowledgments

This work was performed while Wanghong Yuan was at the University of Illinois at Urbana-Champaign. The authors would like to thank Daniel Grobe Sachs for providing the adaptive H263 encoder, other members of the GRACE project for their informative discussions, and the anonymous reviewers and the associate editor,

Professor Mani Srivastava, for their constructive feedback. This work was supported in part by the US National Science Foundation under grants CCR-0205638 and EIA-99-72884. Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the above agencies.

REFERENCES

[1] J. Flinn, E. de Lara, M. Satyanarayanan, D.S. Wallach, and W. Zwaenepoel, "Reducing the Energy Usage of Office Applications," *Proc. Middleware 2001,* pp. 252-63, Nov. 2001.

[2] S. Mohapatra and N. Venkatasubtramanian, "Power-Aware Reconfigure Middleware," *Proc. 23rd IEEE Int'l Conf. Distributed Computing Systems,* May 2003.

[3] S. Gurumurthi, A. Sivasubramaniam, and M. Kandemir, "DRPM: Dynamic Speed Control for Power Management in Server Class Disks," *Proc. 30th Ann. Int'l Symp. Computer Architecture,* pp. 169-179, June 2003.

[4] C. Hughes, J. Srinivasan, and S. Adve, "Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications," *Proc. 34th Int'l Symp. Microarchitecture,* pp. 250-261, Dec. 2001.

[5] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan, "Energy-Adaptive Display System Designs for Future Mobile Environments," *Proc. Int'l Conf. Mobile Systems, Applications, and Services,* pp. 245-258, May 2003.

[6] A.R. Lebeck, X. Fan, H. Zeng, and C.S. Ellis, "Power Aware Page Allocation," *Proc. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS IX),* Nov. 2000.

[7] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," *Proc. Symp. Operating Systems Design and Implementation,* Nov. 1994.

[8] P. Levis et al., "The Emergence of Networking Abstractions and Techniques in TinyOS," *Proc. First Symp. Networked System Design and Implementation (NSDI '04),* Mar. 2004.

[9] H. Zeng, X. Fan, C. Ellis, A. Lebeck, and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* pp. 123-132, Oct. 2002.

[10] B. Li and K. Nahrstedt, "A Control-Based Middleware Framework for Quality of Service Adaptations," *IEEE J. Selected Areas Comm.,* vol. 17, no. 9, pp. 1632-1650, Sept. 1999.

[11] M. Mesarina and Y. Turner, "Reduced Energy Decoding of MPEG Streams," *Proc. SPIE Multimedia Computing and Networking Conf.,* Jan. 2002.

[12] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker, "Agile Application-Aware Adaptation for Mobility," *Proc. 16th Symp. Operating Systems Principles,* pp. 276-287, Dec. 1997.

[13] C. Efstratiou, A. Friday, N. Davies, and K. Cheverst, "A Platform Supporting Coordinated Adaptation in Mobile Systems," *Proc. Fourth IEEE Workshop Mobile Computing Systems and Applications,* pp. 128-137, June 2003.

[14] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," *Proc. 18th IEEE Real-Time Systems Symp.,* pp. 298-307, Dec. 1997.

[15] J. Lorch and A. Smith, "Operating System Modifications for Task-Based Speed and Voltage Scheduling," *Proc. First Int'l Conf. Mobile Systems, Applications, and Services,* pp. 215-230, May 2003.

[16] P. Pillai and K. G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proc. 18th Symp. Operating Systems Principles,* pp. 89-102, Oct. 2001.

[17] V. Raghunathan, P. Spanos, and M. Srivastava, "Adaptive Power-Fidelity in Energy Aware Wireless Embedded Systems," *Proc. IEEE Real Time Systems Symp.,* pp. 106-117, Dec. 2001.

[18] S. Park, V. Raghunathan, and M. Srivastava, "Energy Efficiency and Fairness Tradeoffs in Multi-Resource, Multi-Tasking Embedded Systems," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 469-474, Aug. 2003.

[19] P. Pillai, H. Huang, and K.G. Shin, "Energy-Aware Quality of Service Adaptation," Technical Report CSE-TR-479-03, Univ. of Michigan, 2003.

[20] C. Rusu, R. Melhem, and D. Mosse, "Maximizing the System Value while Satisfying Time and Energy Constraints," *Proc. 23rd Real-Time Systems Symp.,* pp. 246-257, Dec. 2002.

[21] A. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low-Power CMOS Digital Design," *IEEE J. Solid-State Circuits,* vol. 27, pp. 473-484, Apr. 1992.

[22] S. Brandt and G.J. Nutt, "Flexible Soft Real-Time Processing in Middleware," *Real-Time Systems,* vol. 22, no. 1-2, 2002.

[23] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," *Proc. Symp. Operating Systems Principles,* pp. 149-163, Oct. 2003.

[24] R. Liao and A. Campbell, "A Utility-Based Approach for Quantitative Adaptation in Wireless Packet Networks," *Wireless Networks,* vol. 7, no. 5, Sept. 2001.

[25] Y. Hou, H. Tzeng, and S. Panwar, "A Weighted Max-Min Fair Rate Allocation for Available Bit Rate Services," *Proc. IEEE GLOBECOM,* Nov. 1997.

[26] W. Yuan and K. Nahrstedt, "Process Group Management in Cross-Layer Adaptation," *Proc. Multimedia Computing and Networking Conf.,* Jan. 2004.

[27] W. Yuan and K. Nahrstedt, "Integration of Dynamic Voltage Scaling and Soft Real-Time Scheduling for Open Mobile Systems," *Proc. 12th Int'l Workshop on Network and OS Support for Digital Audio and Video,* pp. 105-114, May 2002.

[28] L. Abeni and G. Buttazzo, "Integrating Multimedia Applications in Hard Real-Time Systems," *Proc. 19th IEEE Real-Time Systems Symp.,* pp. 4-13, Dec. 1998.

[29] J. Nieh and M.S. Lam, "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications," *Proc. 16th Symp. Operating Systems Principles,* pp. 184-197, Oct. 1997.

[30] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *Proc. Int'l Symp. Low-Power Electronics and Design,* pp. 197-202, 1998.

[31] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez, "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems," *Proc. 22nd IEEE Real-Time Systems Symp.,* pp. 95-105, Dec. 2001.

[32] L. Yan, J. Luo, and N. Jha, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Heterogeneous Distributed Real-Time Embedded Systems," *Proc. Int'l Conf. Computer-Aided Design,* Nov. 2003.

[33] A. Sinha and A. Chandrakasan, "Dynamic Voltage Scheduling Using Adaptive Filtering of Workload Traces," *Proc. Fourth Int'l Conf. VLSI Design,* pp. 221-226, Jan. 2001.

[34] AMD, *Mobile AMD Athlon 4 Processor Model 6 CPGA Data Sheet,* http://www.amd.com, Nov. 2001.

[35] K. Flautner and T. Mudge, "Vertigo: Automatic Performance-Setting for Linux," *Proc. Symp. Operating Systems Design and Implementation,* pp. 105-116, Dec. 2002.

[36] G. Anzinger et al., "High Resolution POSIX Timers," http://high-res-timers.sourceforge.net, 2004.

[37] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchroniztion," *IEEE Trans. Computers,* vol. 39, no. 9, Sept. 1990.

[38] D. Grunwald, P. Levis, K. Farkas, C. Morrey III, and M. Neufeld, "Policies for Dynamic Clock Scheduling," *Proc. Fourth Symp. Operating System Design and Implementation,* pp. 73-86, Oct. 2000.

[39] T. Pering, T. Burd, and R. Brodersen, "Voltage Scheduling in the lpARM Microprocessor System," *Proc. Int'l Symp. Low Power Electronics and Design,* July 2000.

[40] M. Corner, B. Noble, and K. Wasserman, "Fugue: Time Scales of Adaptation in Mobile Video," *Proc. SPIE Multimedia Computing and Networking Conf.,* pp. 75-87, Jan. 2001.

[41] J. Flinn and M. Satyanarayanan, "Energy-Aware Adaptation for Mobile Applications," *Proc. Symp. Operating Systems Principles,* pp. 48-63, Dec. 1999.

[42] H. Zeng, C. Ellis, A.R. Lebeck, and A. Vahdat, "Currentcy: A Unifying Abstraction for Expressing Energy Management Policies," *Proc. USENIX Ann. Technical Conf.,* pp. 43-56, June 2003.

[43] R. Neugebauer and D. McAuley, "Energy Is Just Another Resource: Energy Accounting and Energy Pricing in the Nemesis OS," *Proc. Eighth IEEE Workshop Hot Topics in Operating Systems (HotOS-VIII),* pp. 67-72, May 2001.

[44] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing Energy and Server Resources in Hosting Centres," *Proc. Symp. Operating Systems Principles,* pp. 89-102, Oct. 2001.

[45] Y.H. Lee, K.P. Reddy, and C.M Krishna, "Scheduling Techniques for Reducing Leakage Power in Hard Real-Time Systems," *Proc. 15th Euromicro Conf. Real-Time Systems,* pp. 105-116, July 2003.

[46] R. Jejurikar and R. Gupta, "Procrastination Scheduling in Fixed Priority Real-Time Systems," *ACM SIGPLAN Notices,* vol. 39, no. 7, July 2004.
[47] R. Jejurikar and R. Gupta, "Energy Aware Task Scheduling with Task Synchronization for Embedded Real Time Systems," *Proc. IEEE Int'l Conf. Compilers, Architecture and Synthesis for Embedded Systems,* pp. 8-11, Oct. 2002.
[48] D. Zhu, R. Melhem, and D. Mosse, "Power Aware Scheduling for AND/OR Graphs in Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 8, pp. 849-864, Aug. 2004.
[49] C. Perez et al., "QoS-Based Resource Management for Ambient Intelligence," *Ambient Intelligence: Impact on Embedded System Design,* pp. 159-182, 2003.
[50] K. Gopalan and T. Chiueh, "Multi-Resource Allocation and Scheduling for Periodic Soft Real-Time Applications," *Proc. SPIE Multimedia Computing and Networking Conf.,* Jan. 2002.
[51] C. Poellabauer, H. Abbasi, and K. Schwan, "Cooperative Run-Time Management of Adaptive Applications and Distributed Resources," *Proc. 10th ACM Multimedia Conf.,* pp. 402-411, Dec. 2002.
[52] E. Lara, D. Wallach, and W. Zwaenepoel, "HATS: Hierarchical Adaptive Transmission Scheduling for Multi-Application Adaptation," *Proc. SPIE Multimedia Computing and Networking Conf.,* Jan. 2002.
[53] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Integrated Power Management for Video Streaming to Mobile Devices," *Proc. ACM Multimedia Conf.,* Nov. 2003.
[54] C. Pereira, R. Gupta, P. Spanos, and M. Srivastava, "Power-Aware API for Embedded and Portable Systems," *Power Aware Computing,* R. Graybill and R. Melhem, eds., pp. 153-166. Plenum/Kluwer, 2002.

Wanghong Yuan received the BS and MS degrees in 1996 and 1999, respectively, from the Department of Computer Science, Beijing University, and the PhD degree in 2004 from the Department of Computer Science, University of Illinois at Urbana-Champaign. Since July 2004, he has been with DoCoMo USA labs, where he is a research engineer. His research interests include operating systems, networks, multimedia, and real-time systems, with an emphasis on the design of energy-efficient and QoS-aware operating systems. He is a member of the IEEE.

Klara Nahrstedt received the BA degree in mathematics from Humboldt University, Berlin, in 1984, and the MSc degree in numerical analysis from the same university in 1985. In 1995, she received the PhD from Dpartment of Computer Information Science at the University of Pennsylvania. She was a research scientist in the Institute for Informatik in Berlin until 1990 and is an associate professor in the Computer Science Department at the University of Illinois at Urbana-Champaign. Her research interests are directed toward multimedia middleware systems, quality of service(QoS), QoS routing, QoS-aware resource management in distributed multimedia systems, and multimedia security. She is the coauthor of the widely used multimedia book *Multimedia: Computing, Communications, and Applications* (Prentice Hall), and she is the recipient of the US National Science Foundation Early Career Award, the Junior Xerox Award, and the IEEE Communication Society Leonard Abraham Award for Research Achievements. She is the editor-in-chief of the *ACM/Springer Multimedia Systems Journal* and she is the Ralph and Catherine Fisher Associate Professor. She is a member of the ACM and a senior member of the IEEE.

Sarita V. Adve received the PhD degree in computer science from the University of Wisconsin-Madison in 1993. She is an associate professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. Her research interests are in computer architecture and systems, with a current focus on power-efficient and reliable systems. She currently serves on the US National Science Foundation CISE advisory committee, served on the expert group to revise the Java memory model from 2001 to 2005, was named a UIUC University Scholar in 2004, received an Alfred P. Sloan Research Fellowship in 1998, IBM University Partnership awards in 1996 and 1997, and an NSF CAREER award in 1995. She was on the faculty at Rice University from 1993 to 1999. She is a member of the IEEE and the IEEE Computer Society.

Douglas L. Jones received the BSEE, MSEE, and PhD degrees from Rice University in 1983, 1985, and 1987, respectively. During the 1987-1988 academic year, he was at the University of Erlangen-Nuremberg in Germany on a Fulbright postdoctoral fellowship. Since 1988, he has been with the University of Illinois at Urbana-Champaign, where he is currently a professor in electrical and computer engineering, the Coordinated Science Laboratory, and the Beckman Institute. He was on sabbatical leave at the University of Washington in Spring 1995 and at the University of California at Berkeley in Spring 2002. In the Spring semester of 1999, he served as the Texas Instruments Visiting Professor at Rice University. He is an author of two DSP laboratory textbooks, and was selected as the 2003 Connexions Author of the Year. He is a fellow of the IEEE and served on the Board of Governors of the IEEE Signal Processing Society from 2002-2004. His research interests are in digital signal processing and communications, including nonstationary signal analysis, adaptive processing, multisensor data processing, OFDM, and various applications such as advanced hearing aids.

Robin H. Kravets received the PhD degree from the College of Computing at the Georgia Institute of Technology in 1999 and is currently an assistant professor in the Computer Science Department at the University of Illinois, Urbana-Champaign. She is the head of the Mobius group at UIUC, which researches communication issues in mobile and ad hoc networking, including power management, connectivity management, transport protocols, admission control, location management, routing, and security. Her research has been funded by various sources, including the US National Science Foundation and HP Labs. She actively participates in the mobile networking and computing community, both through organizing conferences and being on technical program committees. She is currently a member of the editorial board for the *IEEE Transactions on Mobile Computing* and *Elsevier Ad Hoc Networks Journal*. She is also a member of the Steering Committee for WMCSA, the IEEE Workshop on Mobile Computing Systems & Applications. She is a member of the IEEE. For a list of publications and more detailed information, please visit: http://www-sal.cs.uiuc.edu.