To appear in SPIE/ACM Multimedia Computing and Networking Conference (MMCN), 2004

Process Group Management in Cross-Layer Adaptation

Wanghong Yuan, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
{wyuan1,klara}@cs.uiuc.edu

ABSTRACT

Our previous MMCN03 paper reported a cross-layer adaptation framework, GRACE-1, that coordinates the adaptation of CPU frequency/voltage, CPU scheduling, and application quality. GRACE-1 assumes that all application processes (or threads) are independent from each other and adapt individually. This assumption, however, is invalid for multi-threaded applications that include dependent and cooperative processes. To support the joint performance requirements of such dependent processes, this paper extends GRACE-1 with a process group management mechanism. The enhanced framework, called *GRACE-grp*, introduces a new OS abstraction, *group control block*, to provide the OS-level recognition and support of process groups. Through a group control block, dependent processes can explicitly set up a group and specify their dependency in the OS kernel. Consequently, GRACE-grp schedules and adapts them in a synchronized and consistent manner, thereby delivering joint performance guarantees. We have implemented and evaluated the GRACE-grp framework. Our experimental results show that compared to GRACE-1, GRACE-grp provides better support for the joint quality of dependent processes and reduces CPU energy consumption by 6.2% to 8.7% for each process group.

Keywords: Adaptation, Process Group Management, Energy Saving, Quality of Service.

1. INTRODUCTION

Mobile devices, primarily running multimedia applications, are becoming an integral part of the pervasive computing environment. Such mobile systems need both to provide application quality of service (QoS) and to save energy in the presence of application and resource variations. Researchers have therefore introduced QoS- or energy-aware adaptation into different system layers, such as the hardware, OS, middleware, and applications.[1–4] The Illinois GRACE project is developing a *cross-layer adaptation* framework,[5] in which all system layers are adaptive and cooperate for a system-wide optimal objective, e.g., maximizing the overall application quality under all resource constraints.

Our previous MMCN03 paper[6] reported the first GRACE prototype, called GRACE-1, that coordinates CPU frequency/voltage scaling in hardware, soft real-time CPU scheduling in OS, and process quality adaptation in multimedia applications. GRACE-1 uses a hierarchy of *global* and *local* adaptations. A global adaptation responds to large variations (e.g., process joining or leaving) and mediates the hardware, OS, and application layers to maximize the overall application quality under CPU and energy constraints. A local adaptation, on the other hand, handles small variations (e.g., changes of process CPU usage) either to stabilize application quality or to save more energy.

GRACE-1 assumes that all application processes are independent from each other and hence execute and adapt individually. This assumption, however, does not hold for general multi-threaded multimedia applications, such as video-on-demand (VoD), that consist of multiple cooperative processes. These cooperative processes depend on each other and cooperate to process media streams. A VoD client, for example, typically includes two dependent processes, a receiver and a player. These two processes have a producer-consumer relationship: the receiver retrieves data from a video server, while the player decodes the received data. We refer to such a set of dependent processes as a *process group*. Processes in a group present joint performance and resource demands. For the above VoD client example, the packet rate of the receiver and the frame rate of the player should be consistent; otherwise, their shared data buffer would underflow or overflow.

Treating cooperative processes independently, GRACE-1 may result in two problems: (1) *Conflicting adaptation*. During global adaptation, GRACE-1 may choose conflict quality configuration for cooperative processes; e.g., it may increase the receiver's packet rate but decrease the player's frame rate, thereby causing the shared buffer to overflow. (2) *Harmful isolation*. During soft real-time scheduling, GRACE-1 preempts a process with exhausted CPU allocation for temporal and performance isolation. Such isolation inside a process group, however, may potentially compromise the

joint group performance. For example, if the VoD receiver is preempted while in a critical section, the player will also be blocked to wait for the critical section (as explored by Niz et al.[7]).

To address the above two problems, we enhance GRACE-1 with a *process group management* mechanism. The enhanced framework is called *GRACE-grp*. Its major goal is to support the joint quality requirements of dependent processes during cross-layer adaptation. To do this, GRACE-grp provides the OS-level recognition and support of process groups. Specifically, it introduces a new OS abstraction, *group control block* (GCB), that extends the traditional OS process management to support the group behavior of cooperative processes. Similar to the way a process control block interacts with the OS on behalf of its process, a GCB interacts with the GRACE resource manager on behalf of the processes in the group. First, cooperative processes set up a group by creating a GCB in the OS and specifying the group dependency (e.g., the consistency among the member QoS parameters) in the GCB. Second, during global adaptation, GRACE coordinates the quality configuration of dependent processes based on the group dependency, thus adapting their quality consistently. Third, during scheduling, the CPU scheduler executes processes in the group by sharing their allocation, thus avoiding harmful isolations inside a group (but still isolating different groups). Finally, the scheduler also monitors the CPU usage of each process group and triggers local adaptations to handle the usage variations.

This paper makes two major contributions. First, we introduce the GCB abstraction to recognize and support the process cooperation in the OS. The introduction of GCB helps to reduce the gap between the dependency of application processes and the process management of the operating system. Although some other OS abstractions, such as resource container,[8] activity,[9] and reservation domain,[10] provide similar group-aware resource management, GCB differs from them for two reasons: (1) it contains the application semantics (e.g., group dependency) and coordinates the adaptation and execution of cooperative processes; and (2) it is a light-weight approach and requires small modification to the existing operating system. Our current implementation of GRACE-grp, including GCB, adds only 778 lines of C code into the Linux kernel. Second, we have implemented the GRACE-grp framework and experimentally evaluated it on an HP laptop with a variable speed processor and multi-threaded codec applications. Our experimental results show that (1) GRACE-grp provides better support for the joint quality of multimedia applications (e.g., adapting the quality of cooperative processes in a user-preferable manner) than GRACE-1, and (2) for each process group, GRACE-grp saves the CPU energy by 39% to 46% compared to non-adaptive CPU scenario and by 6.2% to 8.7% compared to GRACE-1.

The rest of the paper is organized as follows. First, Section 2 introduces system models used by GRACE-grp. Section 3 describes the group control block. Section 4 discusses how GRACE-grp uses the group control block for group-aware cross-layer adaptation. Section 5 describes the implementation and experimental evaluation. Section 6 compares our approach with related work. Finally, Section 7 summarizes this paper.

2. SYSTEM MODELS

In this section, we extend our previous adaption models[6] to consider process dependency.

Adaptive Processor. We consider mobile devices with a single adaptive processor that supports a discrete set of frequencies, $\{f_1, ..., f_{max}\}$. For a CMOS-based processor, the operating frequency is proportional to the supply voltage; the dynamic power, which dominates the processor energy consumption, is proportional to the frequency and the square of the voltage; i.e., $p(f) \propto fV_f^2$, where $p(f)$ and V_f are the processor power and supply voltage, respectively, at the frequency f.[4] A lower frequency hence enables a lower voltage and saves power and energy.

Adaptive Periodic Process. We consider each multimedia application as a set of *periodic processes*, which manipulate (e.g., compress or transport) media streams. Each process accepts an input and generates an output to other processes or the user. In doing so, it delivers a certain QoS level, which is a vector of application-level QoS parameters, such as rate and delay. To deliver such a QoS level, the process demands a certain amount of CPU resource. *Adaptive* processes support multiple QoS levels $\{q_1, ..., q_m\}$, trading off output quality for CPU demand.[6, 11] For example, a VoD player can decode video with different frame rates and resolutions. Figure 1 illustrates a generic adaptive process model.

During runtime, each process releases a job (e.g., a frame decoding) every period. Each job has a soft deadline, typically defined as the end of the period, and consumes some cycles for its execution. For different jobs, the process needs different amount of cycles due to the changes in the data (e.g., I, P, and B frames or scene changes). We therefore characterize the CPU demand of a process with a period $P(q)$ and a stochastic number of cycles $C(q)$. The ratio of $\frac{C(q)}{P(q)}$ is the process's CPU bandwidth demand (cycles per second). The period can be mapped from the operating QoS level. The parameter

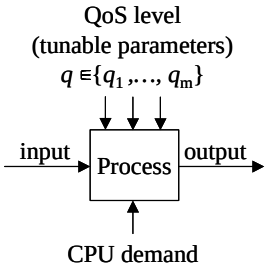

Figure 1. Adaptive process model.

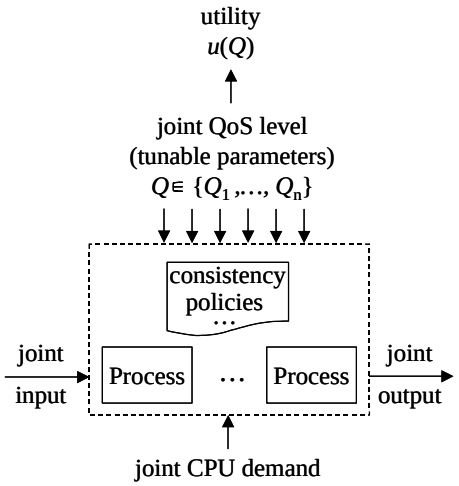

Figure 2. Adaptive group model.

$C(q)$ is the number of cycles a process job demands statistically (e.g., the average or the 95^{th} percentile across all jobs), which can be profiled off-line[12] or online.[13]

Adaptive Process Group. Multi-threaded multimedia applications consist of dependent processes that cooperate with each other to manipulate data streams. We refer to such a set of dependent processes, running in the same device, as a *process group*[*]. The group dependency is application-specific and hence is generally provided by application developers or users. Since this paper concentrates on the supporting mechanisms for group management, we do not formally define the application-specific group dependency. Instead, we illustrate it with two examples:

- *Producer-consumer dependency*. The receiver and player of a VoD client share a buffer, through which the receiver delivers its output as the input of the player. Assume the QoS level of the receiver is packet rate and packet size, and the QoS level of the player is frame rate and frame size (data bits per frame). During the QoS configuration, the receiver and player need to meet the constraint $packet\ rate \times packet\ size = frame\ rate \times frame\ size$; otherwise, the shared buffer may overflow or underflow. During runtime, if the receiver (or player) enters a critical section to write (or read) the buffer, the other one must wait for the critical section.
- *Priority dependency*. A hyper-video player consists of multiple processes to simultaneously play videos that are connected together through hyper-links. The user may focus on one video and dynamically switch her focus to another one. For example, a video tour of a museum can display different floors. When the user clicks a floor, a new window pops up to display the clicked floor. The user can focus one floor when she finds something interesting. The group dependency here is that the focused video should be played with a desirable quality, while others can be played with a lower quality (e.g., a lower frame rate or resolution).

From a group point of view, dependent processes accept a joint input and deliver a joint output. They correspondingly operate at a joint QoS level Q, which is a combination of the QoS level of each member process; i.e., $Q = (q_1, ..., q_m)$, where there are m members, each with QoS level $q_j, 1 \leq j \leq m$. We refer to all possible joint QoS levels that the process group supports as the *group QoS space* $\{Q_1, ..., Q_n\}$. This space is not a simple cross-product of QoS levels of member processes. Instead, all joint QoS levels in the space meet the group dependency (as illustrated in the above two examples).

Each joint QoS level in the space is associated with a utility $u(Q)$, which measures the joint group quality from the view point of other groups or the user.[1, 11, 14] Here, we define the utility for a process group, rather than for individual processes, since dependent processes deliver a joint output quality. Further, we use *system utility* to measure the overall application quality and define it as the weighted sum of utilities of all concurrent groups in the system, i.e., $\sum_{i=1}^{n} w_i u_i$, where there are n groups, each with weight w_i and utility u_i. Figure 2 illustrates a generic group model.

[*]We currently assume that each process belongs to at most one group. This assumption is reasonable for mobile devices since they often run one group or several groups without shared processes (e.g., two groups, a VoD client and a calendar synchronization) at a time.

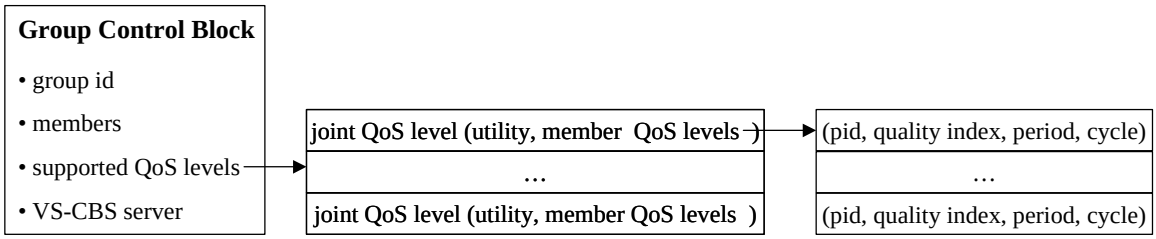

Figure 3. Data structure of a group control block.

Table 1. Operations (system calls) on a group control block.

Purpose	*System call*	*Description*
Group Setup	CreateGroup	Create a GCB in the operating system kernel for a process group.
	JoinGroup	Join a group and become a member in the GCB.
	LeaveGroup	Leave a group. The GCB is destroyed after all members have left.
	SetJointQoS	Add a joint QoS level, which meets the group dependency, for the process group.
	GetProcessQoS	Get the coordinated QoS level for a process.
Group-aware Scheduling	RequestGrpRes	Request CPU allocation for a process group.
	ReleaseGrpRes	Release CPU allocation of a process group.
	SetProcessPriority	Set a process's priority within the group. The priorities of member processes define their execution order.

3. GROUP CONTROL BLOCK: A NEW OS ABSTRACTION

Since cooperative processes present joint quality and resource demands, they need to interact with the OS as an integrated unit, rather than individually. To support this need, we introduce a new OS abstraction, called *group control block* (GCB), for each process group. The introduction of the GCB extends the traditional process management to provide the OS-level recognition and support of a process group. Similar to the way a process control block interacts with the OS on behalf of its process, the GCB interacts with the OS on behalf of the processes within the group. Specifically, the GCB allows dependent processes to explicitly set up a group in the OS kernel and to tell the OS about their group dependency. As a result, the OS can treat them as a process group, rather than as independent processes.

Figure 3 illustrates a GCB's data structure, which contains four major attributes:

- *Group id*. It identifies the GCB and hence the process group in the operating system.
- *Members*. This attribute contains information about each member process in the group, such as the process id.
- *A list of supported QoS levels*. This attribute is specified by member processes based on the group dependency. Each item in the list corresponds to a supported joint QoS level, including a utility and a list of member QoS. Each member QoS item describes a member's QoS level, including its process id, index to the QoS level, and CPU demand.
- *VS-CBS server*. It is used to schedule processes within a group according to the group dependency. This server mechanism enables group members to share allocation but isolates different groups (see Section 4.3).

Application processes can manipulate the GCB in the OS via a set of system calls (Table 1). They use these calls to set the values of the above GCB attributes. The first five calls are used for group setup and the last three are for scheduling.

In the next section, we describe how to use the GCB to manage application dependency in the GRACE cross-layer adaptation framework. Note that although in this paper, the GCB is used for the coordinated adaptation, it can be generalized for other scenarios, such as group communication and hierarchial group management.

4. GROUP MANAGEMENT IN CROSS-LAYER ADAPTATION

The GRACE-grp framework consists of five major components: a coordinator, a hierarchical CPU scheduler, a CPU adaptor, a set of group control blocks (OS-level group managers), and a set of application-level group managers, as shown in Figure 4. The application-level group manager is responsible both for checking the quality consistency of cooperative

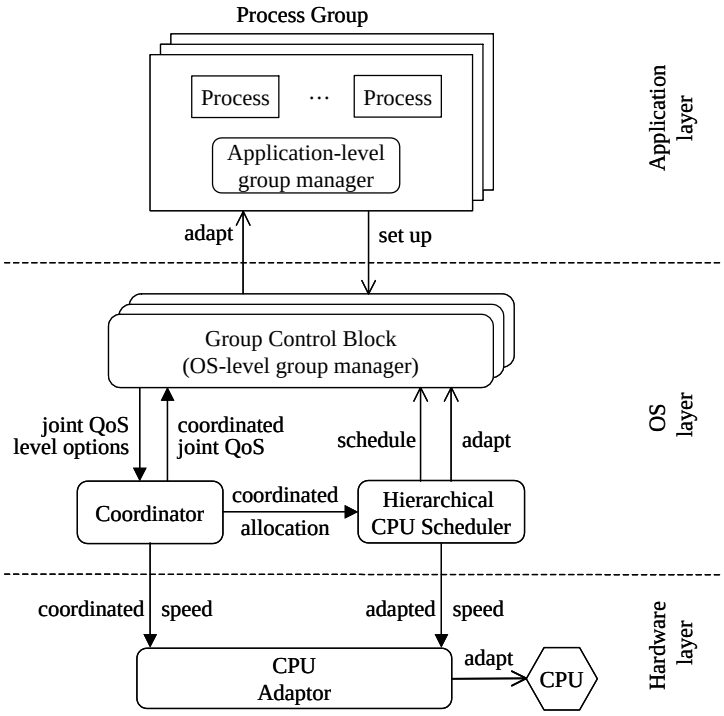

Figure 4. Architecture of the GRACE-grp framework.

processes and for defining other group policies (e.g., how many members and the joining order of members). In the rest of the paper, we do not discuss the application-level group manager in detail since it is application-specific. The group control block of a process group acts as the OS-level group manager and interacts with the coordinator and scheduler on behalf of the processes in the group. The coordinator performs global adaptation in response to large and long-term variations; it mediates the hardware, OS, and application layers to maximize the overall application quality and to save energy. To make this global optimization, the coordinator considers the utility and CPU demand of each group, the CPU speed and power, and the dependency of each group. The scheduler employs a hierarchical real-time scheduling to schedule groups and processes, thus delivering joint quality to each group. It also monitors the runtime CPU usage of individual groups and may trigger local adaptations to handle the CPU usage variations.

Operationally, the group-aware cross-layer adaptation is achieved via a combination of four major operations: group setup, coordination, scheduling, and adaptation. We next describe these operations in detail.

4.1. Group Setup

A set of dependent processes form a group through application semantics (such as the producer-consumer relationship) in the application level. For joint performance and resource requirements, these dependent processes need to express the group dependency in the system level. The current operating system support of group dependency (e.g., via process inheritance and critical sections) is insufficient, since it does not (1) support group-aware adaptation, and (2) control the scheduling of dependent processes. In contrast, the introduction of the GCB enables dependent processes to explicitly set up a group in the OS kernel. We next describe the group setup protocol:

1. The group leader process (e.g., the producer) calls `CreateGroup` to create a group. Upon this call, the operating system creates a GCB in the kernel, adds the calling process to the member list of the GCB, and returns the group id.
2. The group id is passed to other cooperative processes via IPC mechanisms (such as shared memory and message). These processes then calls `JoinGroup` by specifying the group id and are consequently added to the member list of the specified GCB.
3. The group leader process calls `SetJointQoS` to set all possible joint QoS levels the group supports. As a result, the GCB has all necessary information on member processes, such as process id, quality levels, and CPU demand.

4. The group leader process calls `RequestGrpRes` to tell the OS resource manager that the process group enters real-time mode and requires resource guarantees. Upon this call, the coordinator performs a global adaptation to determine the quality configuration and resource allocation for all groups (see Section 4.2).
5. When a member process leaves the group by calling `LeaveGroup`, the OS updates the group control block (e.g., removing the calling process from the member list and updating the joint group QoS levels). This process may also call `RequestGrpRes` to update the resource demand of the group.
6. Finally, if all member processes have left the group or if a process calls `DeleteGroup`, the group control block releases the CPU allocation of the group and is then destroyed.

Note that the problems on how to select the group leader, how to define the joining order of members, and how to find the supported joint QoS levels are handled by the application-level group manager, which is out of the scope of this paper. For example, the supported joint QoS levels can be either statically defined by application developers or dynamically configured by middleware services (e.g.,[2, 15]) during runtime.

4.2. Group-Aware Global Coordination

A global coordination happens upon the system call `RequestGrpRes` and `ReleaseGrpRes` that requests and releases the CPU resource for a process group, respectively. In such a case, the coordinator mediates the hardware, OS, and application layers to maximize the system utility and save CPU energy. Specifically, the coordination problem is to find a joint QoS level for each group and a speed for the CPU, such that (1) the system utility is maximized under the CPU resource constraint, and (2) the CPU speed is minimized for the maximal system utility. That is, we employ a *utility-greedy* approach that first maximizes the system utility and then minimizes the CPU speed and energy consumption.

More formally, assume that (1) there are n groups, each with a weight w_i and group QoS space $\{Q_{i1}, Q_{i2}, ..., Q_{im_i}\}$, (2) each joint QoS level $Q_{ij} = (q_{ij1}, ..., q_{ijt_i})$ in the space has a utility $u_i(Q_{ij})$, and (3) each process QoS level q_{ijk} in the joint QoS level has a stochastic CPU demand of $C(q_{ijk})$ cycles per period $P(q_{ijk})$. We can represent the global coordination as the following constrained optimization problem, which is NP-hard but can be solved efficiently using well-known optimization algorithms, such as dynamic programming.[14]

$$\text{maximize} \quad \sum_{i=1}^{n} w_i u_i(Q_{ij}) \qquad \text{(system utility)} \tag{1}$$

$$\text{minimize} \quad f \in \left\{ f : f \geq \sum_{i=1}^{n} \sum_{k=1}^{t_i} \frac{C(q_{ijk})}{P(q_{ijk})} \right\} \qquad \text{(CPU speed/energy)} \tag{2}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \sum_{k=1}^{t_i} \frac{C(q_{ijk})}{P(q_{ijk})} \leq f_{max} \qquad \text{(EDF schedulability)} \tag{3}$$

$$Q_{ij} = (q_{ij1}, ..., q_{ijt_i}) \in \{Q_{i1}, Q_{i2}, ..., Q_{im_i}\} \quad i = 1, 2, ..., n \tag{4}$$

$$f \in \{f_1, ..., f_{max}\} \tag{5}$$

where Equation (3) is the CPU resource constraint to guarantee the schedulability of the earliest deadline first (EDF) based scheduling algorithm, which will be discussed in Section 4.3.

The above global coordination is similar to that in GRACE-1[6]: they both seek to maximize utility and save energy. However, the difference is that GRACE-grp treats dependent processes as a group. In particular, its coordinator interacts with the group control block to configure process quality, thus making the QoS configuration of dependent processes consistent. This group-aware coordination simplifies the work of application development and reduces the overhead of process QoS configuration. For example, without GCB, dependent processes may need to renegotiate with the coordinator several times for the consistent QoS configuration in the group.

4.3. Group-Aware Hierarchical Scheduling

After the coordinator configures a joint QoS level for each group, the scheduler performs soft real-time scheduling to deliver performance guarantees. The scheduler enables processes within a group to share CPU allocation, thus meeting

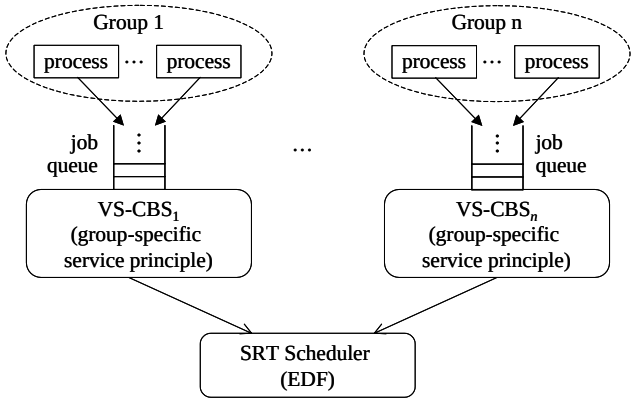

Figure 5. Hierarchical group-aware scheduling.

the group dependency, but provides temporal and performance isolation among different groups. To do this, GRACE-grp employs a hierarchical scheduling scheme (Figure 5), similar to other hierarchical scheduling techniques.[16–19]

Specifically, for each group, we add a variable speed constant bandwidth server (VS-CBS) into its GCB. The VS-CBS is extended from the CBS[20] for a variable speed environment.[19] It is characterized by a maximum budget C and a period P, which are calculated as follows. If the served group is configured with a joint QoS level $Q = (q_1, ..., q_m)$, where each member is configured with a QoS level q_i and is allocated $c(q_i)$ cycles every period $P(q_i)$, then

$$P = \min\{P(q_i) : i = 1, 2, ..., m\} \tag{6}$$

$$C = P \times \sum_{i=1}^{m} \frac{C(q_i)}{P(q_i)} \tag{7}$$

where $\sum_{i=1}^{m} \frac{C(q_i)}{P(q_i)}$ is the total CPU bandwidth demand of the group. Since the server period is equal to the minimum of the process periods, the server can meet the most demanding timing constraint in the group. Each server also has a deadline d and a budget c. As the server executes a job, its budget is decreased by the number of cycles the job consumes. Whenever the budget c is decreased to 0, the budget is recharged to C and the deadline is postponed as $d = d + P$.

The scheduler maintains all VS-CBSs and schedules them on an EDF basis— always selecting the server with the earliest deadline. The selected server then executes jobs, released by process members, according to its service discipline. Different groups can define different service disciplines (i.e., execution order of processes in the group) via system call `SetProcessPriority`. For example, a producer-consumer group can use a priority-based discipline, where the process in a critical section inherits the priorities of other processes waiting for the critical section.[7] The hyper-video group may first serve the focused video and then serve other unfocused ones in a round-robin manner.

To ensure a soft schedulability†, the scheduler needs to validate that the total CPU utilization of all groups is no more than one; i.e., the aggregate CPU bandwidth demand of all groups is below the coordinated CPU performance, as illustrated in Equation (2) and (3). With this schedulability constraint, the VS-CBS based scheduling algorithm provides temporal isolation among different servers (and hence different groups), as analyzed in.[19] Further, it allows resource sharing within a group. Specifically, the VS-CBS server contains all CPU allocation of processes in a group by aggregating their bandwidth demands; it consumes the allocation whenever a member process is executed. That is, VS-CBS server is the resource principal of the group, to which the CPU resource is allocated and against which the CPU usage is charged. In that sense, the VS-CBS is similar to a resource container.[8] However, a VS-CBS differs from a resource container in that, in addition to resource allocation and charging, it also defines how to execute processes in the service principle.

4.4. Group-Aware Adaptation

Multimedia processes often experience large variations in CPU usage due to data changes; e.g., an MPEG decoder process needs different amount of cycles to decode I and B frames. A process may therefore overrun or underrun its allocation,

† *Soft schedulability* means that each process meets deadline statistically, rather than meeting all deadlines as in hard real-time systems.

i.e., need more or less cycle budget[‡]. The above soft real-time scheduling algorithm can efficiently handle scenarios when some processes underrun and others in the group overrun by sharing allocation within a group. However, when all process members overrun or underrun simultaneously, e.g., due to a scene change, it may either degrade the joint quality (with high deadline missing) or waste energy (with CPU idling). We therefore propose a local OS adaptation to handle these overruns and underruns as follows.

In addition to executing processes, a VS-CBS server also monitors the CPU usage of processes in the group and detects if the group needs more or less CPU cycles than allocated. In particular, it counts the number of cycles the served group has consumed, C^s, in each super period, P^s, which is the least common denominator of member periods. The server then uses an exponential average strategy, which is commonly used in control systems, to adapt its CPU allocation. Specifically, it adjusts the maximum budget to

$$C = \alpha \times C + (1 - \alpha) \times \frac{C^s}{P^s} P \tag{8}$$

where α is a tunable parameter and represents the relative weight between the current maximum budget and the observed budget demand.

When the VS-CBS server of a group updates its maximum budget, the total CPU demand of all groups changes. The CPU adaptor hence needs to adjust the speed based on the total CPU demand. If the CPU speed required by the above local OS adaptations is beyond the range of the supported speeds, then the scheduler can (1) tolerate some deadline misses or resource waste or (2) trigger a quality adaptation in the application level. In the latter case, the scheduler interacts with the group's GCB to select another joint QoS level based on the currently available CPU resource. The scheduler then upcalls each process in the group (e.g., via a signal) to adapt the process's QoS parameters. This group-aware quality adaptation ensures that the QoS levels of dependent processes still meet the group consistency after adaptation.

5. IMPLEMENTATION AND EVALUATION

5.1. Implementation

We have implemented a prototype of GRACE-grp on an HP Pavilion N5470 laptop. This laptop has an AMD Athlon processor, which supports six different speeds {300, 500, 600, 700, 800, 1000 MHz}.[21] The operating system is Redhat Linux 7.2 with a modified kernel 2.4.18. This modification adds 778 lines of C code. Specifically, we add two new modules into the kernel, one for speed setting and one for soft real-time (SRT) scheduling. The speed setting module changes the processor speed by writing the frequency and corresponding supply voltage to a system register `FidVidCtl`.[21]

The SRT scheduling module combines the global coordinator and the hierarchical soft real-time scheduler, and is responsible for group management, global coordination, and scheduling. First, we add eight new system calls (see Table 1), which allow multimedia processes to communicate with the OS kernel for group setup and quality adaptation. Second, we add a periodic UTIME timer[22] into the kernel and attach the SRT scheduler as the callback function of the timer. The resolution of the UTIME timer is one ms. Consequently, the SRT scheduler is invoked every ms and performs the two-level hierarchical scheduling. It sets the real-time priority for all multimedia processes and then invokes the standard Linux scheduler, which, in turn, dispatches multimedia processes in the order of their priorities.

To support the two-level hierarchial scheduling, the SRT scheduler uses two kinds of priorities, group priority and relative priority. This is similar to the priority levels in the Windows NT operating system.[23] The group priority is determined by the deadline of the group's VS-CBS server: the earlier the server deadline, the higher the group priority. The relative priority of a process is the process's priority within the group and is set by the process via system call `SetProcessPriority` based on the group dependency (e.g., in the hyper-video player, the focused process has the highest relative priority in the group). Finally, the real-time priority (`rt_priority`) of a process, which is used by the standard Linux scheduler, is the sum of the above two priorities. We make the difference among group priorities large enough so that processes in different groups have different real-time priorities (i.e., processes in the earliest-deadline group have the highest real-time priorities). Using this priority mechanism, the SRT scheduler effectively isolates different groups and allows budget sharing within a same group.

[‡]Although the CPU scheduler allocates budget to a group together, such a group allocation is based on the bandwidth demand of individual processes. Equivalently, each process is allocated $C(q)$ cycles per period $P(q)$.

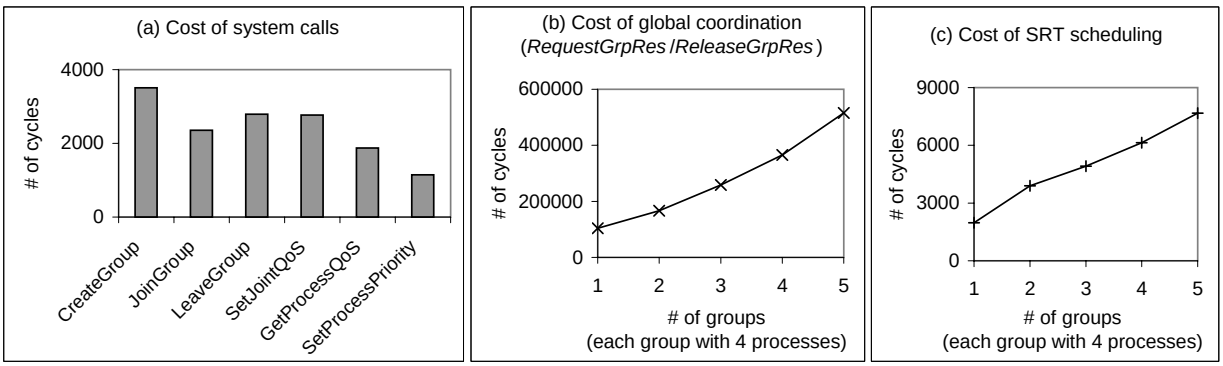

Figure 6. Overhead of the new system calls, global coordination, and soft real-time scheduling.

5.2. Experimental Evaluation

We now present the experimental results to evaluate the GRACE-grp framework. Our previous GRACE-1 work[6] has demonstrated the value of the cross-layer adaptation by comparing cross-layer adaptation with systems that exploit adaptations in one or two layers. This paper therefore focuses on evaluating the benefits of the process group management for multi-threaded multimedia applications, primarily by comparing GRACE-grp with GRACE-1.

5.2.1. Experimented Applications

The experimental workload includes two multi-threaded multimedia applications, a hyper-video player and a secure video sensor. The hyper-video player consists of multiple processes to simultaneously play MPEG videos, which are connected together through hyper-links.[24] Specifically, we use meta-data to associate the link with the frame number of a video. When a video link is available (i.e., the associated frame is being decoded), a new process is started to play the linked video. The user can focus on one video and switch the focus to another one. Each individual MPEG video player, based on the Berkeley MPEG tools, can adapt the frame rate and dithering method (i.e., in `color` or `gray`), trading off video quality and computation. The group dependency here is that the focused video should be played with the highest quality, while others can have lower quality.

The secure video sensor consists of two processes, an encoder and an encrypter, which have a producer-consumer dependency. The encoder encodes images into H263 frames and writes them to a buffer, while the encrypter reads frames from the buffer and encrypts them using the advanced encryption standard (AES). Both processes are adaptive: the H263 encoder, based on the TMN codec, can adapt the DCT (discrete cosine transform) compression and motion search for each frame encoding, trading off compression ratio and computation[25]; the AES encrypter can use different encryption keys of 128, 192, and 256 bits, trading off security and computation. The encoder and encrypter also support two different frame rates, 6 and 10 frames per second (fps). The group dependency here is that the frame rates of the encoder and encrypter need to be the same and their execution is synchronized to avoid the overflow and underflow of the shared buffer.

For each of the above hyper-video player and secure video sensor, we define a set of group QoS levels meeting the group dependency. For each group QoS level, we profile the number of cycles that a member process consumes for a frame processing and use the 95^{th} percentile across all frames as the process's CPU demand. We then define the utility of the group QoS level as follows: assume the group QoS level is $Q = (q_1, ..., q_m)$, where q_i is the i^{th} process's QoS level and is associated with profiled CPU demand of $C(q_i)$ cycles every period $P(q_i)$. The utility for the group QoS level is $u(Q) = \sum_{i=1}^{m} \log \frac{C(q_i)}{P(q_i)}$. Intuitively, the higher the CPU demand, the higher the utility.

5.2.2. Overhead

First, we analyze the overhead of the GRACE-grp framework by measuring the cost for the new system calls in Table 1 and for the hierarchical soft real-time scheduling. In particular, we measure the cost in cycles, rather than processing time, since the latter depends on the speed, while the former is roughly constant with the speed. We run one to five hyper-video players, each with four concurrent MPEG players. For each run, we measure the number of cycles elapsed during each new system call in the application and measure the number of cycles for each SRT scheduling in the kernel 10000 times. We repeat the experiment six times to calculate the average cost. Figure 6 plots the results.

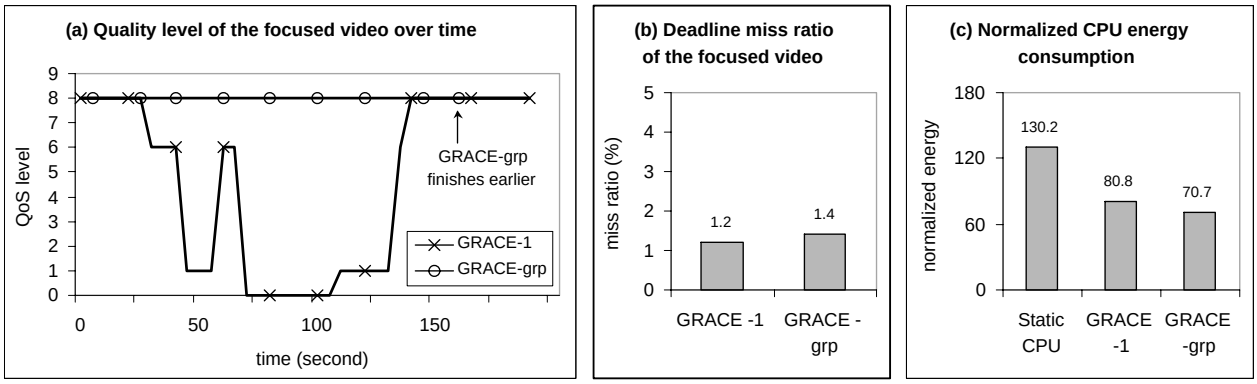

Figure 7. Comparision of GRACE-grp and GRACE-1 for the hyper-video player.

We notice that except `RequestGrpRes` and `ReleaseGrpRes`, the overhead of other system calls is independent of the number of groups and is below 4000 cycles (i.e., no more than 4 microseconds at speed 1 GHz). This is small relative to multimedia processing (e.g., an MPEG frame decoding takes 5.7 to 10 million cycles). On the other hand, the cost of `RequestGrpRes` and `ReleaseGrpRes` depends on the number of groups and is quite large, since they trigger global coordination to reallocate CPU resource among groups. Their costs hence represent the overhead of the global coordination, which is below 6×10^5 cycles (less than 10% of a MPEG frame decoding). This overhead is acceptable since these two system calls are invoked infrequently, i.e., only when a group requests, updates, or releases CPU resource.

Finally, the overhead of soft real-time scheduling is small, e.g., below 9000 cycles for 20 concurrent processes in five groups. The scheduling overhead depends on the number of groups (and hence the number of processes), since the SRT scheduler needs to check the status for each group (e.g., if the member processes begin a new period).

5.2.3. Comparison with GRACE-1

Next, we evaluate GRACE-grp's capability for quality support and energy saving by comparing it with GRACE-1, which adapts and schedules individual processes independently.[6] Since we currently do not have power meters, we evaluate energy consumption with a new metric, *normalized energy*, which is commonly used in the literature.[4, 26] In particular, we normalize the CPU power using the AMD document,[21] where the relative CPU power is 0.22, 0.35, 0.47, 0.6, 0.74, and 1.0 for the speed 300, 500, 600, 700, 800, and 1000 MHz, respectively. If the CPU runs for T time units, its normalized energy consumption is

$$\int_0^T p(f(t))\, dt \tag{9}$$

where $f(t)$ and $p(f(t))$ are the CPU speed and relative power, respectively, at time t, $0 \leq t \leq T$.

Run the hyper-video player. We first run a hyper-video player under GRACE-grp. The hyper-video player consists of four MPEG players; each supports nine QoS levels, with CPU bandwidth demand from 114 to 336 million cycles per second. We assume the first video is the user focused one; this video has 4000 frames with frame size 352×240 pixels, and its frame 150, 300, and 450 have a link to a video with 2000 frames. Consequently, the second, third, and fourth MPEG players will start at the frame number of 150, 300, and 450, respectively. After the hyper-video player runs for 25 seconds, we start a synthetic application, which demands 500 million cycles per second and lasts for 10 seconds. The purpose of this synthetic application is to trigger global coordination.

During this experiment, we measure the coordinated quality level and deadline miss ratio of the first video (i.e., the user-focused one), as well as the normalized CPU energy during the runtime of the hyper-video player. We repeat the experiment five times to calculate the average of the three metrics. Finally, we do the same experiment under GRACE-1, which ignores the group dependency and adapts and schedules each individual MPEG player independently.

Figure 7-(a) plots the changes of the QoS level of the focused video. It is obvious that in GRACE-1, the quality of the focused video varies largely over time, since GRACE-1 treats the focused video equally as other concurrent videos. On

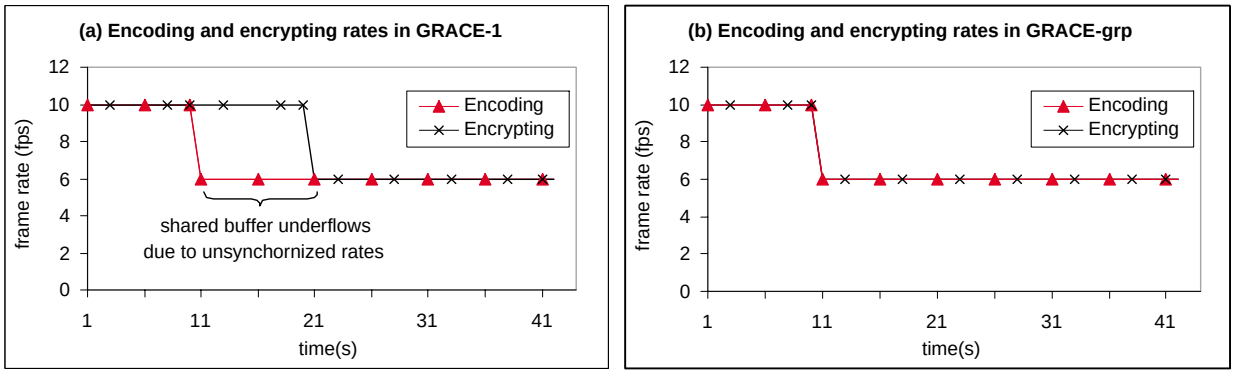

Figure 8. Coordinated frame rates of the encoding and encrypting in GRACE-grp and GRACE-1.

the other hand, GRACE-grp always configures the focused video with the highest quality level 8 (with dithering method `color2` and frame rate 25 fps). The reason is that in the supported group QoS levels, the focused video always has the highest quality configuration; consequently, when the CPU resource is insufficient, GRACE-grp degrades the quality (and hence CPU demand) of other unfocused videos to meet the group dependency.

Figure 7-(b) shows the deadline miss ratio of the focused video. GRACE-grp and GRACE-1 miss 1.4% and 1.2% of jobs deadlines, respectively. Recall that in Section 5.2.1, we use the 95^{th} percentile of the profiled cycles as the process CPU demand; therefore, about 5% of process jobs overrun the allocation and may potentially miss the deadline. GRACE-grp and GRACE-1 both reduce the deadline miss ratio effectively, but use different approaches. GRACE-1 allocates an extra CPU budget to an overrun job to avoid the deadline miss.[6] In contrast, GRACE-grp shares the allocation in the group; the focused video can first utilize the shared allocation, since it has the highest relative priority in the group.

The advantage of GRACE-grp's budget sharing over GRACE-1's budget adaptation is that GRACE-grp consumes less energy, as shown in Figure 7-(c). Although GRACE-grp and GRACE-1 both save energy compared to the static CPU scenario when the CPU always runs at the highest speed, GRACE-grp saves more energy. The reason is that unlike GRACE-1 that changes the CPU speed for overrun or underrun jobs, GRACE-grp keeps the speed relatively constant. This smooth speed setting reduces the CPU energy consumption due to the convex nature of the speed-energy function.[4] Another reason for GRACE-grp's energy saving is that it runs the focused video at the highest quality (with the highest frame rate); consequently, the hyper-video player finishes earlier than in GRACE-1, as illustrated in Figure 7-(a).

Run the secure video sensor. We next run the secure video sensor to encode and encrypt 300 images under GRACE-grp and GRACE-1. Again, we run a synthetic application to trigger global coordinations. This synthetic application starts at time 10 seconds, demanding 200 million cycles per seconds, and changes the demand to 400 million cycles per second at time 20 seconds. We measure the frame rate of the H263 encoder and AES encrypter of the video sensor. Recall that the group dependency in the video sensor is that the encoder and encrypter need to have the same frame rate; otherwise, the shared buffer would overflow (resulting in frame loss) or underflow (resulting in long delay).

Figure 8 shows the results. As we can see, GRACE-1 may cause conflicting adaptation of dependent processes. For example, in the time interval 10 to 20 seconds, the encoder has a lower rate (6 fps) than the encrypter (10 fps), thus causing the shared buffer to underflow. In contrast, GRACE-grp coordinates the adaptation of dependent processes, such that their QoS levels still meet the group dependency after adaptation. In particular, it decreases the rates of the encoder and encrypter simultaneously at time 10 seconds, and decreases their bandwidth demands (without changing rates) simultaneously at time 20 seconds. These results clearly indicate the benefits of GRACE-grp's group-aware global adaptation.

Now, we run the video sensor without resource competition. In this case, GRACE-grp and GRACE-1 achieve the same coordination for the encoder and encrypter, both of which operate at the highest QoS levels with the rate 10 fps and with the highest CPU bandwidth demand. We measure the energy consumption and the jitters of the frame encoding and encrypting. Ideally, the length of the time interval between adjacent frame encoding/encrypting should be 100 ms since the frame rate

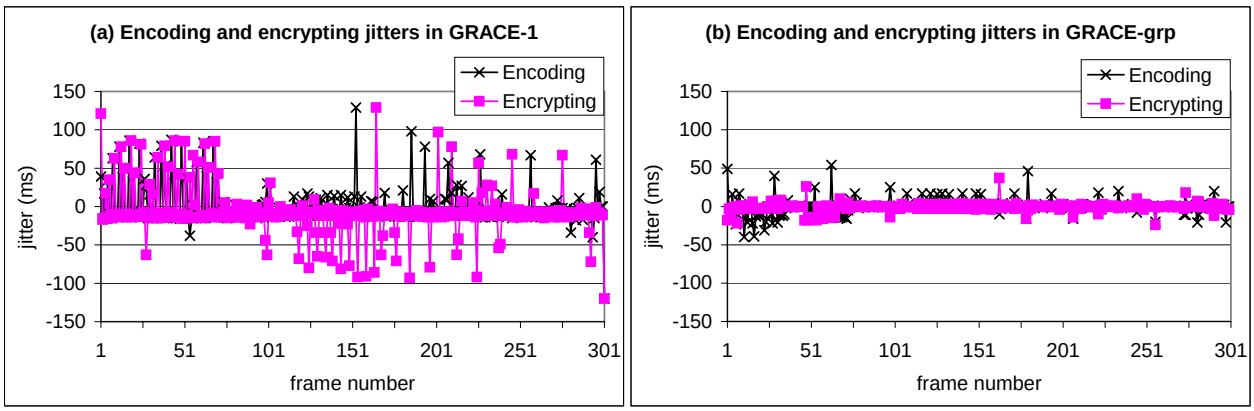

Figure 9. Runtime performance and energy consumption of the secure video sensor in GRACE-grp and GRACE-1.

Table 2. Comparison of energy consumption in GRACE-grp and GRACE-1.

	Static CPU	GRACE-1	GRACE-grp
Normalized CPU energy	49.4	37.1	35.2

is 10 fps. The actual time interval length, however, changes dynamically, since different frames need different number of CPU cycles. The jitter is the difference between the actual interval length and the ideal interval length (100 ms). The results in Figure 9 shows that compared to GRACE-1, GRACE-grp achieves smaller jitters both for frame encoding and for frame encrypting. That is, the video sensor encodes and encrypts images more smoothly. The reason is that GRACE-grp allows dependent processes in a group to share their resource allocation to handle CPU usage changes.

Finally, Table 2 shows the energy consumption of GRACE-grp, GRACE-1, and static CPU scenario. Same as the case of the hyper-video player run, GRACE-grp and GRACE-1 both save energy relative to the static CPU scenario. However, GRACE-grp saves more energy than GRACE-1 since GRACE-grp sets the CPU more smoothly.

6. RELATED WORK

Coordinated adaptation. Mobile multimedia systems need both to support application QoS and to save the battery energy. Researchers have therefore introduced QoS- or energy-aware adaptations into different system layers, such as hardware,[4, 27] operating system,[3] middleware,[2, 28] and applications.[1] Recently, there have been several research contributions on the coordination of adaptations of multiple applications or different system layers.[2, 17, 27, 29, 30] For example, the ECOSystem[27] manages energy as a first class resource and adapts application energy allocation for a specified battery lifetime. Q-fabric[30] is a set of kernel-level abstractions, that can combine cooperative application adaptation with distributed resource management. Our previous work[5, 6] reported the GRACE-1 framework, where all system layers are adaptive and cooperate with each other to maximize the system utility under all resource constraints. Most of the above work implicitly assumes that application processes are independent from each other and adapt individually. The GRACE-grp framework presented in this paper is built on our previous GRACE-1 work,[6] but adds group management for dependent processes in multi-threaded applications. It provides the OS-level recognition and support for cooperative groups of processes and coordinates the adaptation and execution of dependent processes for a desirable joint quality.

Hierarchial scheduling. Another closely related work includes hierarchial scheduling and budget sharing.[7, 16, 18] For example, the H-CBS[18] partitions application processes into subsets (groups) and schedules them hierarchically to support both process and group performance. Niz et al.[7] extended priority inheritance and priority ceiling protocol techniques to achieve temporal isolation when processes share common resources (e.g., critical section). The above related work is orthogonal and complementary to the soft real-time scheduling in the GRACE-grp framework; e.g., the priority inheritance protocol[7] can be used to set the process dependency in a group. GRACE-grp is different from the above related work for two reasons. First, its soft real-time scheduling is integrated with the cross-layer adaptation: the resource allocation

is determined by the global coordinator and the scheduler also dynamically adapts resource allocation during runtime. Second, the soft real-time scheduling is energy-aware: it saves energy while providing soft performance guarantees.

OS support for process groups. Group management has been extensively studied in the network domain (e.g., for group communication). In the OS domain, abstractions similar to the group control block (GCB) include activity, reservation domain, and resource container. An activity in Rialto[9] accounts for resource consumption for a collection of processes across the OS protection domain. A reservation domain in Eclipse[10] is a collection of processes and corresponding resource reservations. The basic idea behind reservation domains is to protect performance isolation among reservation domains. A resource container[8] encompasses all system resources that a server uses to perform a particular independent activity, e.g., serving a client connection. The major goal of resource containers is to separate the resource principle from protection domain for fine-grained resource management. These related abstractions focus on the resource management for different entities (i.e., activities, domains, or containers); their major purpose is to isolate and protect among different entities. In contrast, the GCB is primarily applied in the cross-layer adaptation and extends the traditional OS process management. It provides the OS-level recognition and support of process groups. With the GCB, the GRACE-grp framework can coordinate the adaptation of dependent application processes and synchronize their execution for a joint quality. Furthermore, unlike the above OS abstractions, the GCB is a light-weight approach and requires only small modification to the existing OS, as demonstrated in Section 5.

7. CONCLUSION

Multimedia applications typically initiate multiple processes (or threads), which need to cooperate with each other to transport, display, and otherwise manipulate media streams. On a mobile device, the cooperation of group processes aims not only for timeliness and synchronization, but also for energy-efficient resource usage to extend the battery lifetime. This paper presents a group-aware cross-layer adaptation framework, called *GRACE-grp*, that enhances our previous GRACE-1 framework[6] with a group mechanism for dependent processes in multi-threaded multimedia applications. The primary goal of GRACE-grp is to deliver a user-desirable joint quality for dependent processes in the cross-layer adaptation. To do this, GRACE-grp introduces a new operating system abstraction, *group control block*, to provide the OS-level recognition and support for process groups. A group control block acts as the OS-level group manager and interacts with the OS resource manager on behalf of the member processes. Consequently, GRACE-grp coordinates the quality adaptation and scheduling of dependent processes to make their quality configuration and runtime performance consistent. Our experimental results demonstrate that compared to GRACE-1 that adapts and schedules processes individually, GRACE-grp provides better support for the joint quality of cooperative processes and saves energy by 6.2% to 8.7% for each group.

Our future work will extend the group-aware cross-layer adaptation to a distributed environment and study flexible group adaptation policies. For example, when the secure video sensor uses wireless communication, it is desirable to adapt the encoder for a high compression ratio if the wireless bandwidth is low, and to adapt the encrypter for a high security level if the transmission zone is insecure.

ACKNOWLEDGMENTS

We would like to thank Daniel Sachs for providing the adaptive H263 encoder and other members of the GRACE project, especially Professor Sarita Adve, for their informative discussions on energy saving and feedback on process group management. This work was supported by the NSF grant under CCR 02-05638 and CISE EIA 99-72884. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

REFERENCES

1. M. Corner, B. Noble, and K. Wasserman, "Fugue: time scales of adaptation in mobile video," in *Proc. of SPIE Multimedia Computing and Networking Conference, San Jose, CA*, Jan. 2001.
2. S. Mohapatra and N. Venkatasubtramanian, "Power-aware reconfigure middleware," in *Proc. of IEEE 23nd intl. Conference on Distributed Computing Systems*, May 2003.
3. B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker, "Agile application-aware adaptation for mobility," in *Proc. of the 16th Symposium on Operating Systems Principles*, Dec. 1997.

4. M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. of Symposium on Operating Systems Design and Implementation*, Nov. 1994.
5. S. Adve *et al.*, "The Illinois GRACE Project: Global Resource Adaptation through CoopEration," in *Proc. of Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN02), New York City, NY*, June 2002.
6. W. Yuan *et al.*, "Design and evaluation of a cross-layer adaptation framework for mobile multimedia systems," in *Proc. of SPIE Multimedia Computing and Networking Conference*, Jan. 2003.
7. D. Niz, L. Abeni, S. Saewong, and R. Rajkumar, "Resource sharing in reservation-based systems," in *Proc. of 22nd IEEE Real-Time Systems Symposium*, Dec. 2001.
8. G. Banga, P. Druschel, and J. Mogul, "Resource containers: A new facility for resource management in server systems," in *Proc. of 3rd Symposium on Operating System Design and Implementation*, Feb. 1999.
9. M. Jones, D. Rosu, and M. Rosu, "CPU reservations & time constraints: Efficient, predictable scheduling of independent activities," in *Proc. of 16th Symposium on Operating Systems Principles*, Oct. 1997.
10. J. Bruno, E. Gabber, B. Ozden, and A. Silberschatz, "The Eclipse operating system: Providing quality of service via reservation domains," in *Proc. of USENIX 1998 Annual Technical Conference, Berkeley, CA*, June 1998.
11. S. Brandt, "Performance analysis of dynamic soft real-time systems," in *Proc. of the 20th IEEE International Performance, Computing, and Communications Conference (IPCCC 2001)*, Apr. 2001.
12. B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," in *Proc. of 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, Dec. 2002.
13. W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," in *Proc. of Symposium on Operating Systems Principles (SOSP'03)*, Oct. 2003.
14. C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, "A scalable solution to the multi-resource qos problem," in *Proc. of the 20th IEEE Real-Time Systems Symposium*, Dec. 1999.
15. X. Gu and K. Nahrstedt, "Dynamic QoS-aware multimedia service configuration in ubiquitous computing environments," in *Proc. of IEEE 22nd Intl. Conf. on Distributed Computing Systems (ICDCS 2002)*, July 2002.
16. Z. Deng and J. S. Liu, "Scheduling of real-time applications in an open environment," in *Proc. of 18th IEEE Real-time Systems Symposium*, Dec. 1997.
17. E. Lara, D. Wallach, and W. Zwaenepoel, "HATS: hierarchical adaptive transmission scheduling for multi-application adaptation," in *Proc. of SPIE Multimedia Computing and Networking Conference*, Jan. 2002.
18. G. Lipari and S. Baruah, "A hierarchical extension to the constant bandwidth server framework," in *Proc. of 7th IEEE Real-Time Technology and Applications Symposium, Taipei, Taiwan*, May 2001.
19. W. Yuan and K. Nahrstedt, "Integration of dynamic voltage scaling and soft real-time scheduling for open mobile systems," in *Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, May 2002.
20. L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proc. of 19th IEEE Real-Time Systems Symposium, Phoenix, AZ*, pp. 4–13, Dec. 1998.
21. AMD, "Mobile AMD Athlon 4 processor model 6 CPGA data sheet." http://www.amd.com, Nov. 2001.
22. KURT, "UTIME— micro-second resolution timers for linux." http://www.ittc.ku.edu/kurt, Mar. 2003.
23. D. Solomon and M. Russinovich, *Inside Microsoft Windows 2000*, Microsoft Press, 2000.
24. J. W. Jackson, "Multiprocessor soft real time CPU resource manager and its validation with hypermedia applications." Master Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2002.
25. D. Sachs, S. Adve, and D. Jones, "Cross-layer adaptive video coding to reduce energy on general-purpose processors," in *Proc. of IEEE International Conference on Image Processing*, Sept. 2003.
26. P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. of 18th Symposium on Operating Systems Principles*, Oct. 2001.
27. 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.
28. J. Flinn, E. de Lara, M. Satyanarayanan, D. Wallach, and W. Zwaenepoel, "Reducing the energy usage of office applications," in *Proc. of Middleware 2001, Heidelberg, Germany*, Nov. 2001.
29. C. Efstratiou, A. Friday, N. Davies, and K. Cheverst, "A platform supporting coordinated adaptation in mobile systems," in *Proc. of 4th IEEE Workshop on Mobile Computing Systems and Applications*, June 2003.
30. C. Poellabauer, H. Abbasi, and K. Schwan, "Cooperative run-time management of adaptive applications and distributed resources," in *Proc. of 10th ACM Multimedia Conference*, Dec. 2002.