The International Journal of Time-Critical Computing Systems, 15, 249–273 (1998)

Criticality- and QoS-Based Multiresource Negotiation and Adaptation

J. HUANG* huang@htc.honeywell.com
Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN 55418

P.-J. WAN* pwan@htc.honeywell.com
Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN 55418

D.-Z. DU** dzd@cs.umn.edu
Department of Computer Science, University of Minnesota, Minneapolis, MN 55455

Abstract. This paper presents design, analysis, and implementation of a multiresource management system that enables criticality- and QoS-based resource negotiation and adaptation for mission-critical multimedia applications. With the goal of maximizing the number of high-criticality multimedia streams and the degree of their QoS, it introduces a dynamic scheduling approach using on-line QoS adjustment and multiresource preemption. An integrated multiresource management infrastructure and a set of scheduling algorithms for multiresource preemption and on-line QoS adjustment are presented. The optimality and execution efficiency of two preemption algorithms are analyzed. A primal-dual-algorithm-based approximation solution is shown (1) to be comparable to the linear-programming-based solution, which is near optimal; (2) to outperform a criticality-cognitive baseline algorithm; and (3) to be feasible for on-line scheduling. In addition, the dynamic QoS adjustment scheme is shown to greatly improve the quality of service for video streams. The multiresource management system is part of the Presto multimedia system environment prototyped at Honeywell for mission-critical applications.

Keywords: multiresource management, adaptive resource management, QoS negotiation and adaptation, real-time scheduling, optimization and analysis, system prototyping, multimedia system

1. Introduction

We consider a class of continuous multimedia applications that are dynamic and criticality-driven. *Continuous multimedia* comprises video, audio, and image streams, with each having a data flow rate (e.g., 30 frames per second for a video stream). Supporting steady flow of media streams is an essential task of the underlying system resource management services. *Criticality* refers to the importance of multimedia applications. For instance, an application performing periodic image capturing and flaw detection in advanced process control (Guha et al., 1995) can be more important than one that monitors floor activities in the controlled plant, and consequently, the image stream is more critical than the video stream. Therefore, processing such media streams requires that the underlying system services be criticality-cognitive and be able to support more critical multimedia data streams in the presence of multiple service requests. In addition to the criticality-driven nature, the multimedia applications are often *dynamic* and may vary greatly in their demands on system

* This work was supported in part by Rome Laboratory under Contract F30602-93-C-0172 and by the Honeywell Initiatives R&D Program under grant I4560-ME-2000.
** This work was supported in part by the NSF under grant CCR-9530306.

resources. In mission management, for example, detection of a mobile target may trigger a sequence of reactions such as video monitoring, infrared tracking, image library retrieval and target matching and recognition, media data fusion and filtering, and command and control (Robinson, 1993). Such dynamic workloads are not predictable a priori and therefore require applications to negotiate on line for, and adapt to, available system resources, including disk I/O bandwidth, CPU cycles, memory space, video compression/decompression capacity, etc. Without sufficient resources and proper resource management, multimedia streams may lose their data or timeliness in a random fashion, causing application malfunction. We designate this type of criticality-driven, dynamic, and resource-bandwidth-sensitive applications as mission-critical applications.

To support the mission-critical multimedia applications, we have developed and prototyped a multimedia system environment, called P*resto* (Huang et al., 1997c). The multiresource management system is part of the P*resto* that enables quality-of-service (QoS)-based dynamic resource negotiation and adaptation and criticality-based resource preemption. We characterize the applications with three attributes—media stream flow timing, QoS, and criticality—which are orthogonal to each other. Further, we model system resources as "buckets," with each having a capacity limit defined by its scheduling algorithm. The media streams "flow" through the buckets, occupying a certain amount of space in each bucket. The aim of our resource management system is to execute as many high-criticality media streams as possible and at the same time provide the best QoS support, without violating the bucket capacity constraints.

Our approach to this NP-hard multiresource management problem consists of several new concepts. First, a two-phase QoS adjustment scheme is used for allocating resources for a new stream. The first phase of this scheme, called the *shrinking phase*, reduces the QoS of executing streams to accommodate the new stream, achieving the goal of maximizing the number of concurrent streams. The second phase, called the *expansion phase*, expands the QoS of the concurrent streams once the new stream is admitted, achieving the goal of QoS maximization. Second, a criticality-based multiresource preemption scheme is employed in case of resource contention where the system has no sufficient resources to meet the minimum QoS requests. Two approximation algorithms are developed toward the goal of supporting high-criticality applications and maximization of the total number of concurrent applications. Finally, a resource negotiation and adaptation software mechanism is provided to support the on-line QoS adjustment and criticality-based preemption. It enables all the concurrent applications to participate in the negotiation (or re-negotiation) and adaptation process upon a rate, QoS, or criticality change made by any of the applications.

The rest of this paper is organized as follows. In Section 2, we characterize the properties of mission-critical multimedia applications from the user perspective. Section 3 establishes a system resource management infrastructure and associated scheduling algorithms for the individual resources. The core of this paper, namely the QoS-based resource negotiation and adaptation and criticality-based multiresource preemption, is presented in Section 4. We report our system implementation and performance analysis results in Section 5. The related work is discussed in Section 6. We conclude this paper in Section 7.

2. Application Characterization

We characterize the mission-critical multimedia applications according to three factors: timing, quality of service (QoS), and criticality. These factors are specified by application users.

- *Timing*—We consider two parameters regarding the continuous media timing constraints: rate and latency. Rate (λ) is defined in media data units per second, where a unit can be a video frame or a group of audio samples consisting of a certain number of bytes. Latency (L) is the tolerable end-to-end delay from the time the very first media unit is produced at the stream source to the time it reaches the stream destination.

- *QoS*—Quality of service specifies the degree of service quality expected by the application from the underlying computer system. Examples include image resolution, jitter, and so on, which depend largely on application semantics. In our work, we define the QoS as Consecutive Loss Factor (CLF)—the maximum number of consecutive data units allowed to be dropped between every two processed units. As illustrated in Figure 2-1, CLF $= 2$ means that two out of every three units can be dropped (or skipped). Further, the application specifies its CLF using a range $[0, CLF_{max}]$, with 0 being the best case without skipping and CLF_{max} the worst with the maximum number of allowable data units dropped. At run time, the application may adapt its CLF between 0 and CLF_{max} depending on the availability of system resources.

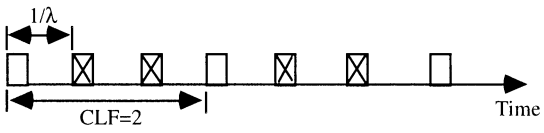

Figure 2-1. QoS definition.

The CLF definition represents a class of QoS semantics, often termed *sieve function* in the context of imprecise computing (Liu et al., 1991). Other examples in such a QoS class are the "leaky bucket" used for network congestion control and the "period overrun" in process control applications, where a control loop may afford to skip a few consecutive periods yet still meeting its control objective.

- *Criticality*—Criticality refers to the degree of application importance among concurrent applications. Application criticality is classified by multiple levels. In general, we consider three classes of applications: "critical," "essential," and "non-essential." As illustrated in Figure 2-2, the critical class requires a guarantee of the minimum QoS of applications. In the essential class, as many applications as possible need to be executed based on their criticality; they may be suspended in case of resource contention. In the non-essential class, as many applications as possible need to be executed; they may be suspended arbitrarily. In our work, the class of critical applications are guaranteed with fixed resource allocation. Hence, our focus is on adaptive resource management for the

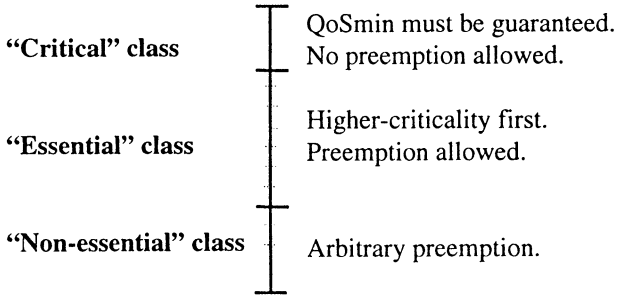

Figure 2-2. Classification of criticality.

essential and non-essential classes of applications with the goal of executing as many as possible higher-criticality applications in the two classes.

As shown in Figure 2-3, the timing, QoS, and criticality factors are orthogonal to each other (i.e., a user may specify any of the parameter values independently of each other). For example, a high-rate application may have low criticality or low QoS requirements, and so on. Our objective is to allocate and schedule the system resources such that the applications' timing constraints are met, QoSs are maximized, and the number of executing (high-criticality) applications are maximized.

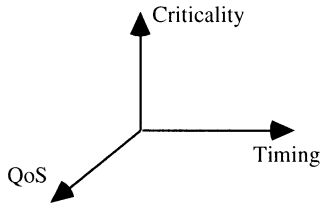

Figure 2-3. The relationship between timing, QoS, and criticality.

We further consider the dynamic behavior of mission-critical applications in terms of workload imposed on the system resources. As illustrated in Figure 2-4, application workload, and in turn resource demand, varies from time to time depending on the operation mode during an application life span (Stankovic & Ramamritham, 1988). A simple example is a sequence of video operation modes (e.g., play, pause, fast forward) with each demanding a certain amount of system resources over a period of time. Every mode change of applications requires allocation/re-allocation of system resources for the requested operation mode. The objective of the resource management is to provide admission control service up on each mode change request and to dispatch the new operation mode for execution if it is schedulable according to its QoS.

3. System Resource Management Architecture

In this section, we briefly discuss the P*resto* resource management architecture initially developed for an end system and the scheduling algorithms used for the individual resources. Multiresource management with QoS negotiation and criticality preemption will be discussed in the next section.

3.1. Session Model

We use the notion of *session* to capture the execution behavior of continuous media applications. From the application perspective, a session corresponds to an operation mode as illustrated in Figure 2-4. From the system perspective, a session consists of producer and consumer threads and a double buffer between the producer and the consumer, according to the real-time producer-consumer paradigm (Jeffay, 1993; Huang & Du, 1994). A session may demand a certain amount of disk I/O bandwidth for storage access, memory space for buffering, CPU cycle for media data processing, and/or video processing bandwidth. From the system resource management point of view, session is the unit of resource allocation and scheduling.

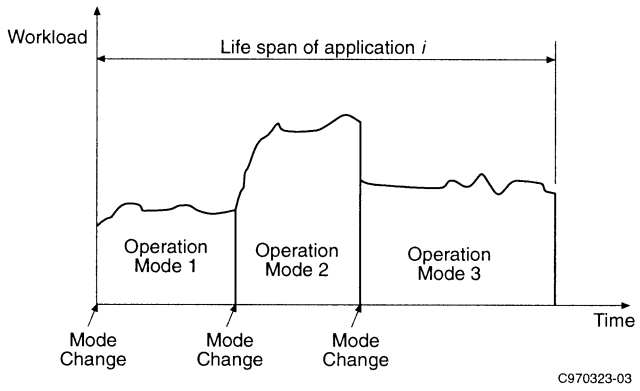

Figure 2-4. Operation modes of application i and their workloads.

Specifically, a session S_i is defined by (λ_i, L_i, $CLFmax_i$, c_i, τ_i, m_i), where

- λ_i, L_i, $CLFmax_i$, c_i—are the stream rate, latency constraint, and consecutive loss factor of QoS and the application criticality as described in Section 2. Note that the actual CLF of a session, denoted by $CLFa_i$, is determined on line during the QoS negotiation (to be discussed in Section 4). Therefore, the actual stream flow rate of the session will be

$$r_i = \frac{\lambda_i}{1 + CLFa_i}, \text{ where } CLFa_i \in [0, CLF\max_i]$$

- τ_i—are the producer and consumer threads with CPU execution time e_i for processing one unit of media data and e_i' for a disk I/O operation. In our work, e_i and e_i' are obtained through "profiling,", i.e., measurement of actual CPU execution time. Another approach can be "compiler-assisted analysis" by tracing code instructions and computing instruction overhead (Niehaus, 1991).

- m_i—is the double buffer allocated to a session. Its size is equal to $(2x_i u_i)$, where x_i is the number of data units processed by either the producer thread or the consumer thread in every execution period and u_i is the size of one data unit. x_i is determined on line by the scheduling algorithms to be discussed below.

3.2. Resource Management Infrastructure

To schedule the application sessions, we employ a three-level resource management approach as shown in Figure 3-1. At the bottom level is a commercial (real-time) operating system. Its function is to provide system primitive services such as setting the priority of a thread and preempting an executing thread. Our design philosophy is to make the P*resto* system open and portable as opposed to inventing yet another operating system.

At the middle level are individual resource schedulers. We decouple the scheduling algorithms and mechanisms of these schedulers: the mechanisms carry out actual scheduling operations (e.g., creating a thread, changing a thread priority, suspending a thread, etc., in case of CPU scheduling), whereas the scheduling algorithms are exercised by a system resource manager for systemwide resource management. With the P*resto* end system, we consider four schedulers for CPU, disk I/O, memory buffer, and video/display window resources, respectively[1].

At the top level is the system resource manager, which allocates system resources on the basis of sessions. It uses the scheduling algorithms of the individual resource schedulers for systemwide schedulability analysis and coordinates the individual schedulers for session execution. As highlighted in the figure, this paper focuses on the system resource manager, discussing its criticality- and QoS-based multiresource scheduling.

3.3. Scheduling of Individual Resources

Before presenting our systemwide resource management approach, let us briefly review the scheduling algorithms used for the individual resources.

CPU Scheduler

As discussed in Section 2, all the threads are periodic in nature. Further, thread access to media data buffers is nonblocking when a double-buffering technique is employed. Thus, we simply adopt the rate-monotonic analysis (RMA) approach (Liu & Layland, 1973) for

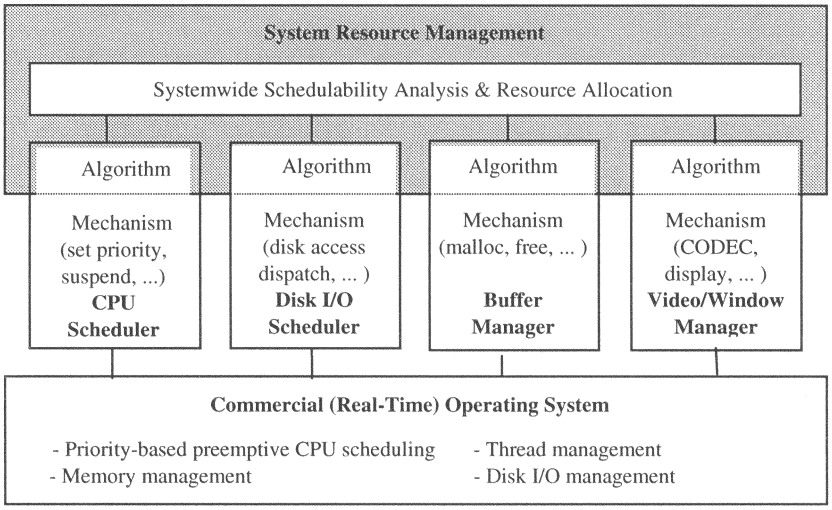

Figure 3-1. Three-level resource management infrastructure.

the CPU scheduling: that is, a number of n sessions are schedulable at the CPU if

$$\sum_{i=1}^{n}(e_i r_i + e_i'/L) \leq \ln 2 \equiv C_{\max} \tag{1}$$

The condition is reasonable when n becomes greater than 5. In practice, n takes a much larger value.

Disk I/O Scheduler

Commercial disk subsystems usually provide I/O scheduling support, often with a SCAN algorithm, at the SCSI controller level. To reduce the disk head movement overhead and guarantee a bounded access time, we employ an interval-based I/O access policy (Huang & Wan, 1996). With the policy, n sessions are schedulable only if

$$\sum_{i=1}^{n} m_i u_i + nSD_{\max} \leq LD_{\max} \tag{2}$$

where L is the latency tolerable by all the n sessions; $D_{\max}$ is the amount of contiguous data that the disk can transfer in one second; S is the disk seek time for serving each I/O request within L, and $m_i = \lceil \lambda_i L \rceil$, which is the number of data units fetched within L.

Other disk I/O scheduling algorithms for admission control can be found in the literature (Gemmell et al., 1995; Vin et al., 1995).

Video and Window Display Scheduler

Under the current P*resto* system, we treat the JPEG video processor and its window display as one "black box" without real-time control. The associated scheduler performs only an admission control function as part of the system resource manager.

The n sessions may deliver video frames at the aggregated rate of $\sum_{i=1}^{n}$. Let $V_{\max}$ be the maximum supportable video rate. Then n sessions can be schedulable if

$$\sum_{i=1}^{n} r_i \leq V_{\max} \tag{3}$$

Buffer Manager

The buffer manager is responsible for admission control as part of the system resource manager. Its operations consists of memory allocation and deallocation using the underlying operating system services. The n sessions consumes $2\sum_{i=1}^{n} x_i u_i$ bytes of memory, where $x_i = \lceil r_i L \rceil$ which is the number of data units processed within L. If the maximum memory space available is $M_{\max}$ bytes, n sessions can be supported if

$$2\sum_{i=1}^{n} x_i u_i \leq M_{\max} \tag{4}$$

Clearly, n sessions are *schedulable* systemwide if Conditions (1), (2), (3), and (4) are met.

Note that instead of developing "yet another" individual resource scheduler, the focus of our work is on integrated multiresource management in a framework where individual schedulers can be "plugged-and-played". As can be observed from Conditions (1), (2), (3), and (4) above, a general formula that we consider for conducting admission control (bandwidth reservation) on an individual resource can be expressed as

Resource consumption of n sessions $\leq$ *resource capacity*

It means that other resource scheduling algorithms or policies may apply to our multiresource management framework as long as they follow the general formula and exhibit linear behavior required by the linear programming approach presented in the next section. For example, the Earliest-Deadline-First (EDF) scheduling formula developed by Liu and Layland (1973) can replace Condition (2) for the CPU admission control. Similarly, a disk I/O scheduling formula developed by Kenchammana-Hosekote & Srivastava (1996) can be used for the disk I/O admission control. Therefore, the individual resource schedulers and their admission control conditions presented in this section are instances under the general multiresource management framework shown in Figure 3-1. The scheduling accuracy depends on the accuracy of individual scheduling algorithms and the tightness of service time supported by the underlying operating system.

Below we discuss the core of our work, i.e., integrated management of the individual resource schedulers to meet sessions' timing, QoS, and criticality requirements.

4. QoS- and Criticality-Based Resource Negotiation and Adaptation

Our approach to the multiresource scheduling problem consists of a scheduling mechanism, a scheduling strategy, and a set of scheduling algorithms. The scheduling mechanism is shown in Figure 4-1. The system resource manager (SRM) maintains a *criticality-ordered waiting queue* for arrival sessions and preempted sessions. The queues associated with the individual resources are managed by the individual resource schedulers. If there are sufficient resources, the system resource manager will dispatch a session for execution. Otherwise, it conducts "automatic QoS negotiation" within the QoS range $[0, CLF_{\max}]$ of the sessions for the available resources. Criticality-based session preemption may take place when a higher-criticality session arrives but there are no sufficient resources after QoS negotiation. The session preemption differs from the thread (or process) preemption in traditional operating systems in that the session is preempted from the multiple resources as opposed to from a single CPU. If a session cannot be scheduled with QoS negotiation and preemption operations, the system resource manager may renegotiate with the application on line for its willingness to lower its QoS specification. This renegotiation is called "interactive QoS negotiation."

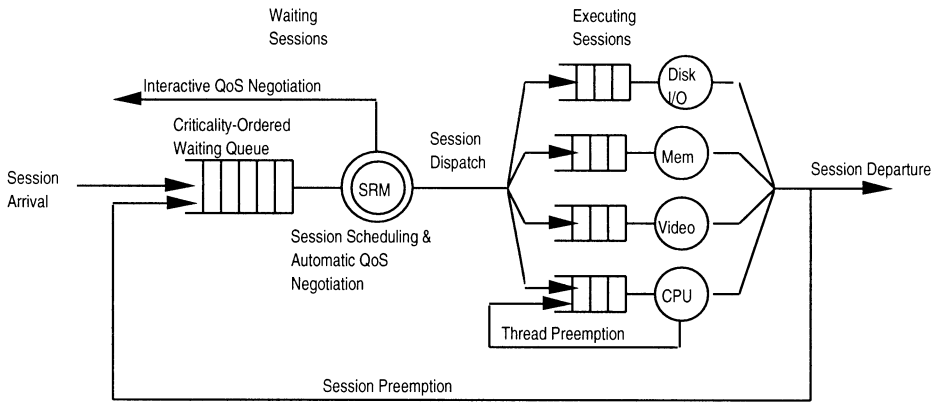

Figure 4-1. Scheduling mechanism.

Our scheduling strategy is illustrated in Figure 4-2. The system resource manager is triggered by either arrival of a new session or departure of a completed session. In general, as highlighted in the diagram, the scheduling process consists of a two-phase QoS adjustment and a session preemption, if necessary. The two-phase QoS adjustment consists of a QoS shrinking phase and a QoS expansion phase. During the shrinking phase, the system resource manager virtually shrinks the QoS of all executing sessions to their minimums (i.e., $CLFa = CLF_{\max}$) to yield the resources to waiting sessions. Its objective is to execute as many waiting sessions as possible. During the expansion phase, the system resource manager tries to increase the QoS of all the executing sessions toward their maximums (i.e.,

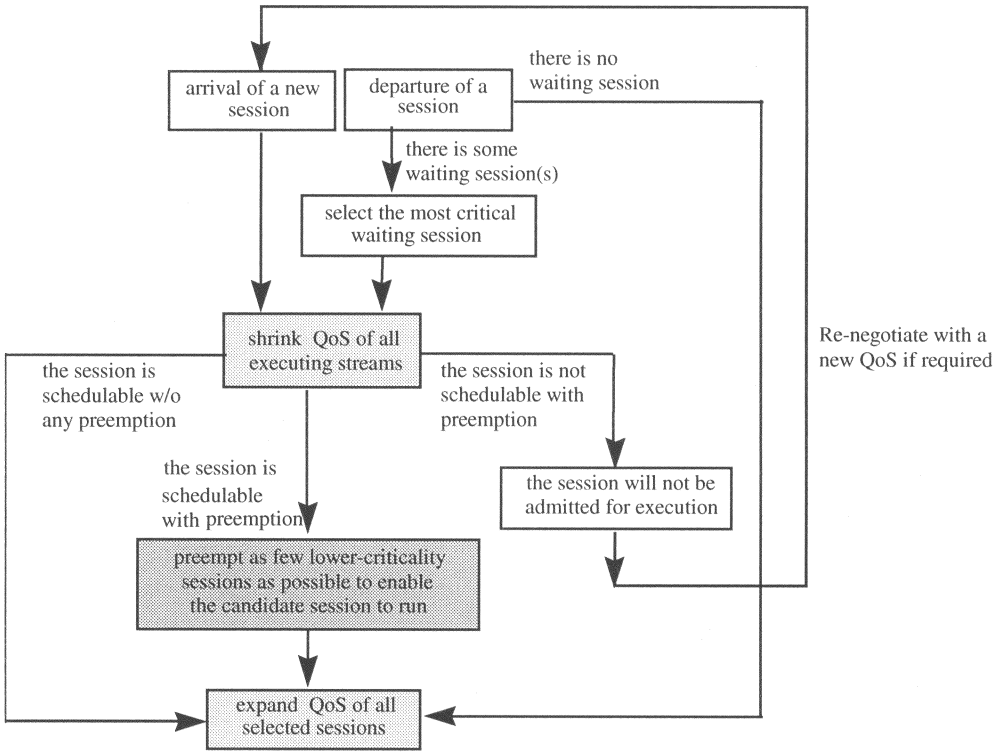

Figure 4-2. The scheduling strategy.

$CLFa = 0$). Its goal is to maximize the QoS of the executing sessions. The preemption of lower-criticality session(s) takes place between the QoS shrinking and expansion phases when a higher-criticality session is not schedulable. The goal is to serve the higher-criticality session while preempting as few sessions as possible.

The set of scheduling algorithms deal with QoS shrinking, session preemption, and QoS expansion. In the following paragraphs, we discuss the design of the algorithms in detail.

(1) QoS Shrinking

Each time we schedule a candidate session in the waiting queue, we first *virtually* reduce the QoSs of all executing sessions to their lowest level, i.e.,

QoS Shrinking:
for $i = 1$ *to* n *do*
    $CLFa_i = CLFmax_i$
Let the CLFa of the candidate session be its $CLF_{\max}$.

Then we check if the new session is executable without any preemption. If so, we expand

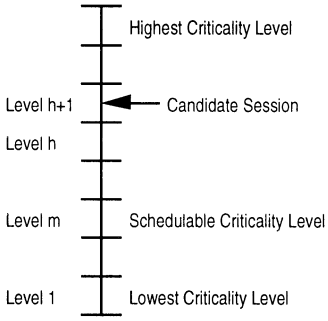

Figure 4-3. Criticality levels within the essential/non-essential class.

the QoS of all executing sessions in the QoS expansion phase. If not, we consider session preemption.

(2) Session Preemption

To conduct session preemption, we classify the executing sessions into criticality levels according to their criticality value. In other words, each criticality level may contain a number of sessions with the same criticality value. As illustrated in Figure 4-3, there can be h different criticality levels below the level of the candidate session being scheduled. Among the h levels, we define the lowest criticality level above which all the executing sessions are still schedulable after insertion of the candidate session as the *schedulable criticality level*. After the insertion of the candidate session, some executing sessions in or below the schedulable criticality level may have to be preempted. For the preemption process, we first need to find the schedulable criticality level and then schedule as many sessions as possible from the levels at or below the schedulable criticality level.

The schedulable criticality level, denoted by m, can be found by a binary search procedure. After we find the schedulable criticality level m, we consider how to support as many sessions as possible from lower m levels. This process is performed in the top-down manner, level by level from the criticality level m to the lowest criticality level 1. Let the remaining CPU, video processor, memory, and disk I/O resources available to sessions at or below level k be denoted by C_{rem}, V_{rem}, M_{rem}, and D_{rem}. Suppose that at level k there are n_k sessions $S_{k,1}, S_{k,2}, \ldots, S_{k,n_k}$. We associate each session $S_{k,j}$ with a 0-1 variable y_{kj} such that session $S_{k,j}$ is scheduled if and only if $y_{kj} = 1$. Then the optimal restoration can be formulated as the following integer linear programming:

$$(IP_k) \qquad \max \quad \sum_{j=1}^{n_k} y_{kj}$$

$$s.t. \quad \sum_{j=1}^{n_k} y_{kj}(e_{kj}r_{kj} + e'_{kj}/L) \leq U_{rem}$$

$$\sum_{j=1}^{n_k} y_{kj} x_{kj} \le V_{rem}$$
$$\sum_{j=1}^{n_k} y_{kj} M_{kj} \le M_{rem}$$
$$\sum_{j=1}^{n_k} y_{kj} (m_{kj} u_{kj} + SD_{\max}) \le D_{rem}$$
$$y_{kj} = 0 \text{ or } 1, \ 1 \le n_k$$

Because this is an NP-hard optimization problem (Huang & Du, 1994), we want to find an approximation solution that is near optimal on one hand and efficient for on-line use on the other hand. Here we consider two solutions; the first is linear programming based add-back and the second is primal-dual based removal. We examine the optimality-efficiency tradeoff of these two solutions through the performance study presented in Section 5.2.

Approximation Solution 1: Linear Programming Based Add-Back

The idea is that at each criticality level we start with *no* scheduled sessions and then add back sessions in some *greedy* order. If there are sufficient resources to schedule a session, the session will be scheduled. Otherwise, we consider the next session. The key to this approach is to determine a good greedy order. In this linear-programming-based "add back," we first find an optimal solution of (LP_k), the linear programming relaxation of (IP_k), in which the integer constraint $y_{kj} = 0$ or 1 is relaxed to the real-number constraint $0 \le y^*_{kj} \le 1$ for $1 \le j \le n_k$. Then the sessions at level k are sorted in the non-increasing order of this optimal solution, which results in the greedy order in which the sessions are added back. The approximation algorithm can be formally described as follows:

LP-Based Approximation Algorithm:
Step 1. Solve (LP_k). *Let* (y_{kj}, $j = 1, \ldots, n_k$) *be an optimal solution.*
Step 2. Order y_{kj} *such that*
$y_{k1^*} \ge y_{k2^*} \ge \cdots \ge y_{kn^*_k}$
Step 3. Add sessions back according to the order obtained from Step 2.

Approximation Solution 2: Primal-Dual Based Removal

A potential drawback of the linear-programming-based heuristic is its significant computation overhead. To improve the speed of the scheduler, we propose another greedy heuristic based on the primal-dual theorem for the linear programming (Fang & Puthenpura, 1996). The idea of this algorithm is that at each criticality level we start with *all* sessions and then remove sessions in some *greedy* order until resource violations disappear. The scheduling of the sessions is performed in the reverse order of the linear-programming-based "add back"

employed by Solution 1, which starts with no sessions and then adds back sessions until resource violation happens. The key to Solution 2 is to find a good greedy order, in which the sessions are removed. In this algorithm, the order is determined in two steps. First, we identify the most critical resource based on some greedy criteria, and then find the session which consumes this critical resource most among all remaining sessions. This session will then be removed (or preempted in terms of scheduling). The selection for identifying the most critical resource is based on the primal-dual theorem for linear programming. To simplify the description of the algorithm, we consider the abstract form of (LP_k):

$$
\begin{aligned}
(P) \quad \max \quad & \sum_{j=1}^{n} x_j \\
s.t. \quad & \sum_{j=1}^{n} a_{ij} x_j \le b_i,\ 1 \le i \le 4 \\
& 0 \le x_j \le 1,\ 1 \le j \le n
\end{aligned}
$$

The dual of the above linear program is as follows.

$$
\begin{aligned}
(D) \quad \min \quad & \sum_{i=1}^{4} b_i y_i + \sum_{j=1}^{n} z_j \\
s.t. \quad & z_j + \sum_{j=1}^{n} a_{ij} y_i \ge 1,\ 1 \le j \le n \\
& y_i \ge 0,\ 1 \le i \le 4 \\
& z_j \ge 0,\ 1 \le j \le n
\end{aligned}
$$

According to the primal-dual theorem, for any primal feasible solution $x_j (1 \le j \le n)$ and any dual feasible solution $y_i (1 \le i \le 4)$, $z_j (1 \le j \le n)$ will imply $x_j = 1$ for any $1 \le j \le n$. Consider the dual feasible solution

$$y_i = 0 (1 \le i \le 4), z_j = 1 (1 \le j \le n)$$

and the corresponding primal solution

$$x_j = 1 (1 \le j \le n)$$

If the primary solution is a primary feasible solution, there is no resource capability violation and all sessions can be scheduled. Otherwise, we will have to remove some session j by setting x_j to 0. The x_j will be selected to minimize the dual objective function by maximally increasing some y_i while keeping others to be 0. To keep the dual solution feasible, the z_j's should be correspondingly decreased and some z_j will vanish because y_i is maximally increased. Suppose y_i is chosen to be increased. Then y_i can be increased by at most:

$$\Delta y_i = \min_{1 \le j \le n} \frac{1}{a_{ij}} = \frac{1}{\max\limits_{1 \le j \le n} a_{ij}}$$

For each $1 \le j \le n$, z_j will decrease by $a_{ij}\Delta y_i$ and z_j will become 0 if a_{ij} achieves $\max_{1\le j\le n} a_{ij}$. Therefore, the dual objective function will be reduced by

$$\sum_{j=1}^{n} a_{ij}\Delta y_i - b_i \Delta y_i = \frac{\sum_{j=1}^{n} a_{ij} - b_i}{\max\limits_{1\le j\le n} a_{ij}}.$$

To maximize this reduction, we will increase y_i such that $\frac{\sum_{j=1}^{n} a_{ij}-b_i}{\max\limits_{1\le j\le n} a_{ij}}$ achieves $\max\limits_{1\le i\le 4} \frac{\sum_{j=1}^{n} a_{ij}-b_i}{\max\limits_{1\le j\le n} a_{ij}}$. Once y_i is found, we will set x_j to 0, where a_{ij} achieves $\max_{1\le j\le n} a_{ij}$ because the corresponding z_j now becomes 0. Based on the above observation, we select the resource i which has the biggest value $\frac{\sum_{j=1}^{n} a_{ij}-b_i}{\max\limits_{1\le j\le n} a_{ij}}$ as the most critical resource. This leads to our second greedy algorithm:

Primal-Dual-Based Algorithm:
Step 1. Let $S = \{1, 2, \ldots, n\}$.
Step 2. While S is not the feasible solution of the primary problem, do
Step 2.1 Find i *such that* $\frac{\sum_{j\in S} a_{ij}-b_i}{\max\limits_{j\in S} a_{ij}}$ *achieves* $\max_{1\le i\le 4} \frac{\sum_{j\in S} a_{ij}-b_i}{\max\limits_{j\in S} a_{ij}}$.
Step 2.2 Find $j \in S$ *such that* a_{ij} *achieves* $\max_{j\in S} a_{ij}$.
Step 2.3 Set $S = S - \{j\}$.

(3) QoS Expansion

Following our scheduling approach illustrated in Figure 4-2, we expand the QoS of all the schedulable sessions at the end of the scheduling. We consider this process to be a policy issue. We first sort all the schedulable sessions in increasing order of QoS and put them in a circular list. Then we expand their QoS in round-robin order.

QoS Expansion:

```
Sort all selected sessions and put them in a circular linked list.
Let S be the first session.
While (the circular list is not empty) do
    If (decreasing S.CLFa by 1 will still satisfy the resource
    constraints) then
        S.CLFa = S.CLFa − 1;
        if (S.CLFa == 0) then remove S from the circular list;
    else
        remove S from the circular list;
    S = S.next;
```

Suppose the maximum of $CLF_{\max}$ is Q and the maximum number of sessions is N, then the QoS expansion procedure will take time $O(\mathrm{QN})$.

5. Performance Analysis and Prototyping

In this section, we examine the performance of the multiresource preemption algorithms and the QoS maximization scheme in terms of their optimality (upper bound), baselines (lower bound), and run-time overhead. This is done through both mathematical analysis and simulation experimented on the P*resto* system platform. We also briefly describe the implementation of the multiresource scheduler.

5.1. Optimality Analysis

We have proposed two approaches to the multiresource preemption problem. As observed from our simulation results, the linear-programming-based approach always performs equally well as or slightly better than the primal-dual-algorithm-based approach with respect to the goal of maximizing the number of concurrent high-criticality multimedia streams. Hence we analyze the optimality of the linear-programming-based approach.

THEOREM *For the linear-programming-based multiresource preemption approach, the difference between the number of sessions restored under our priority assignment and the maximum number of sessions that could be added is at most four. That is, let* y_k^O *be the optimal solution for* (IP_k). *Let* y_k^A *be the solution obtained by the approximation algorithm. Then*

$$\sum_{j=1}^{n_k} y_{kj}^A \geq \sum_{j=1}^{n_k} y_{kj}^O - 4$$

Proof: It follows immediately from the fact that every basic feasible solution of (LP_k) has at most four components that are not integers. This fact can be proved easily. In fact, a feasible solution is basic[2], if and only if it is a vertex in the feasible region. Consider a feasible solution y'_k that has at most five components, say y'_{k1}, y'_{k2}, y'_{k3}, y'_{k4}, y'_{k5}, which are not integers. From the system of equations

$$\sum_{j=1}^{5} z_j (e_{kj} r_{kj} + e'_{kj}/L) = 0$$

$$\sum_{j=1}^{5} z_j x_{kj} = 0$$

$$\sum_{j=1}^{5} z_j M_{kj} = 0$$

$$\sum_{j=1}^{5} z_j m_{kj} u_{kj} + SD_{\max} \sum_{j=1}^{5} z_j x_{kj} = 0$$

we can find a nonzero solution z_1, z_2, z_3, z_4, z_5. Now, we consider point

$$y'_{n_k}(\varepsilon) = (y'_{k1} + \varepsilon z_1, y'_{k2} + \varepsilon z_2, y'_{k3} + \varepsilon z_3, y'_{k4} + \varepsilon z_4, y'_{k5} + \varepsilon z_5, y'_{k6}, \ldots, y'_{kn_k})$$

Because $0 < y'_{kj} < 1$, *for* $1 \le j \le 5$, $y'_{n_k}(\varepsilon)$ is feasible for sufficiently small $|\varepsilon|$. Moreover, $\dfrac{y'_{n_k}(\varepsilon) + y'_{n_k}(-\varepsilon)}{2} = y'_{n_k}$. Therefore, y'_k is not a vertex of the feasible region. ■

In tuition, the theorem indicates that if the optimal solution, which is unachievable in practice, can admit a maximum number of streams, n, then our approximation solution is very close to the optimal one with at least $(n - 4)$ streams admitted.

5.2. Simulation Evaluation

We further compare the two multiresource preemption ("restoration") algorithms with a lower-bound baseline via simulated workloads and system settings on the P*resto* system platform (SPARC20 Station). We consider three system resource schedulers. The first scheduler, denoted LP, is our system resource manager that employs the linear-programming-based approximation optimization algorithm. The second scheduler, PD, employs the primal-dual-algorithm-based approximation optimization algorithm. The third scheduler is a criticality baseline scheduler, CB, in the sense that, like LP and PD, it performs session preemption in the order of increasing criticality levels. However, unlike them, it does not consider optimization while performing preemption within a criticality level. All three schedulers perform both the QoS shrinking and expansion for QoS negotiation.

To compare the proposed two-phase QoS adjustment scheme, we consider a QoS baseline scheduler, denoted as QB, which uses the same approximation algorithm as LP for preemption optimization but does not perform QoS expansion.

All the schedulers are summarized in Table 5-1. Our objective is to understand the effect of our optimized preemption schemes by comparing LP, PD, and CB and the effect of our QoS expansion technique by comparing LP against QB.

Table 5-1. Evaluated resource management schemes.

Scheduler	Criticality-Based Preemption	QoS Shrinking	QoS Expansion
1. LP	Linear-programming-based algorithm	Yes	Yes
2. PD	Primal-dual-based algorithm	Yes	Yes
3. CB (criticality baseline)	Random selection for preemption within each criticality level	Yes	Yes
4. QB (QoS baseline)	Linear-programming-based algorithm	Yes	No

Table 5-2. Workload and system setting specification.

Parameter	Meaning	Default
$\text{Session}_{\text{Max}}$	Total number of sessions in the system	400
C_{Max}	Number of session criticality levels	5
Rate_{min}, Rate_{max}	Session rate uniformly distributed in [Rate_{min}, Rate_{max}]	[30, 30]
CLF_{min}, CLF_{max}	Session QoS uniformly distributed in [CLF_{min}, CLF_{max}]	[0,30]
Latency	Maximum tolerable latency (in seconds) for all the sessions	2
D_{min}, D_{max}	Size of one data unit (JPEG frame) (in Kbytes), uniformly distributed in [D_{min}, D_{max}]	[5, 50]
CPU_{min}, CPU_{max}	CPU time of processing one data unit (in microseconds) uniformly distributed in [CPU_{min}, CPU_{max}]	[2, 3]
CPUIO	CPU time of processing one I/O request (in microseconds)	1
SeekTime	Average disk head seek time between two session streams (in milliseconds)	8
U_{max}	Maximum capacity of the system processor	0.69
M_{max}	Maximum capacity of the system memory (in MBytes)	128
D_{max}	Maximum amount of contiguous data that the disk can transfer in one second (in MBytes per second)	10
V_{max}	Maximum JPEG video processing/display capability (in frames per second)	500

The multimedia application workload and system settings for the simulation runs are defined in Table 5-2. As an advantage of simulation-based performance evaluation, the settings provide us with the flexibility to select and vary a number of parameters, such as the large number of sessions (streams), which are otherwise impractical to generate on the hardware/software platform constrained by the Parallax Video product. Note that the specified stream workload is heterogeneous in the sense that individual streams differ in the actual rate determined by [$\text{Rate}_{\text{actual}} * (\text{CLF}_{\text{actual}} + 1)$]. Thus, it reflects practical workloads in continuous multimedia environments. Further, the system parameter settings are primarily based on the P*resto* system profiling and vendor specifications.

Effect of Criticality

Based Preemption—As shown in Table 5-3, we run our simulation with the workloads of 50, 100, 150, 200, 250, and 300 sessions, respectively. For each workload, we compare the performance of the three schedulers—LP, PD, and CB—in terms of the number of sessions actually scheduled for execution. As indicated in the table, LP, PD, and CB are comparable when the number of sessions in the system is set at 50. This is because there

Table 5-3. Effect of criticality-based session preemption.

Criticality	**Total Number of Sessions vs. Number of Scheduled Sessions**																	
	50			100			150			200			250			300		
	LP	PD	CB	LP	PD	CB	LP	PD	CB	LP	PD	CB	LP	PD	CB	LP	PD	CB
Level 5 (High)	9	9	9	19	19	19	34	34	34	44	44	44	56	56	56	61	61	61
Level 4	7	7	7	20	20	20	24	24	24	28	28	17	6	6	3	0	0	0
Level 3	15	15	15	26	26	21	5	5	2	0	0	0	0	0	0	0	0	0
Level 2	10	10	10	2	2	0	1	1	0	0	0	0	1	1	0	0	0	0
Level 1 (Low)	9	9	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

are sufficient system resources. The effect of session preemption can be observed at 100, where sessions belonging to the lower-criticality levels are preempted. Specifically, the sessions at level 1 are completely preempted under CB. When there are more sessions in the system, only higher-criticality sessions can be executed. Compare LP, PD, and CB at criticality level 4 for 200 sessions: LP and PD scheduled 28 while CB 17. That is, both of our approximate optimization algorithms, LP and PD, scheduled about 60% more sessions than CB, performing significantly better than the baseline approach, which is criticality-cognitive but does random preemption within each criticality level. When the total number of sessions reaches 300, LP, PD, and CB perform the same, admitting 61 sessions. This is understandable, since the system resources are saturated with capacities being able to run sessions only at the highest-criticality level.

The reader may notice that there is a "scheduling anomaly" for the workload of 250 sessions, where both LP and PD algorithms scheduled 1 session from level 2, and none from level 3. To explain the "scheduling anomaly", consider an example, where at the very last scheduling point, there was one session, A, left in Criticality Level 2 and one session, B, in Level 3. At this point, the system resources could accommodate only the resource demand of session A but not that of session B. With the goal of maximizing the number of (higher-criticality) sessions, the system run session A even though its criticality is lower that of session B. Therefore, this result actually demonstrates the correct behavior of the resource management system.

Effect of Algorithm Run-time Overhead

A key issue in developing optimization algorithms for on-line scheduling is algorithm computation overhead. Consider a single live video stream with the data rate of 30 frames per second. To be able to capture the live stream, the system scheduling operation must be completed in less than 33 milliseconds (ms). Hence, an algorithm with a high overhead is not feasible for scheduling multimedia streams, no matter how good the algorithm is in terms of optimality. In this experiment, we compare the computation overheads of the

three schedulers: LP, PD, and CB. The experiment is conducted on a Sun SPARCstation 20/Solaris 2.5. Due to lack of a precise system clock to measure CPU time, we use the Sun Solaris 2.5 Trace Normal Form (TNF) Utilities (SunSoft, 1995) to measure the elapsed time between start and end of each algorithm computation with the precision of 1 microsecond.

As shown in Figure 5-4, the LP computation time increases sharply as the number of sessions being optimized increases, whereas PD and CB have relatively low computation overheads. When the number of sessions is 60, LP's computation time is 126.85 ms, PD's is 2.76 ms, and CB's is 0.26 ms. It is especially interesting to compare LP and PD—the two approximation optimization algorithms—using Table 5-1 and Figure 5-4: LP and PD are comparable in terms of optimality. However, PDís overhead is significantly lower than LPís and is reasonable for practical use.

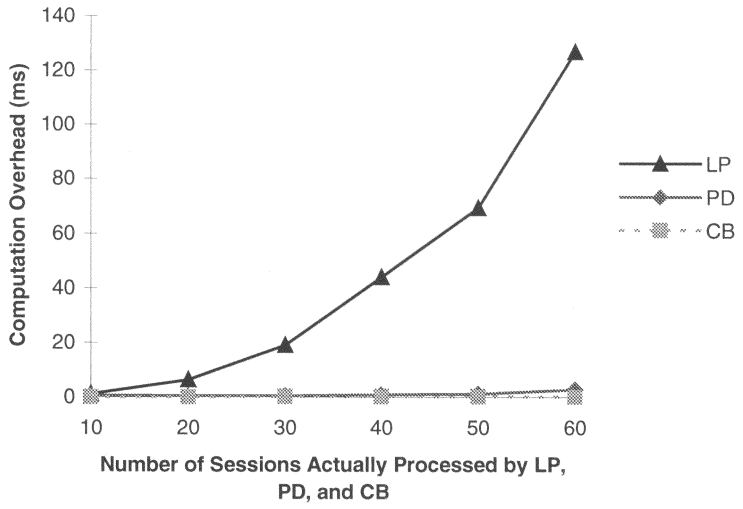

Figure 5-4. Computation overhead of the three preemption algorithms.

Note that the actual number of sessions processed by each of the three algorithms presented by the x-axis is lower than the total number of sessions in the system. There are two reasons for this. First, the system has 5 criticality levels. For a total of 400 sessions with equal distribution of application criticality across the five levels, each of the algorithms deals with at most 80 sessions at each level. Second, in our algorithm implementation, we add a preprocessing stage to reduce the number of sessions to be processed by the approximation algorithms. Therefore, the algorithms deal with only a relatively small number of sessions while making the preemption ("restoration") decision.

Effect of QoS-Based Resource Negotiation and Adaptation

Next we examine the effect of the QoS expansion mechanism employed by the system resource manager as described in Section 4. In particular, we compare LP with QB—a baseline approach that does not increase the QoS of the scheduled sessions even if there are some "left-over" resources. We define a performance metric, called *Accumulated QoS Improvement (AQI)*, as:

$$AQI = \sum_{i=1}^{C_{\max}} \sum_{j=1}^{n_i} (CLF_{\max_{ij}} - CLFa_{ij})$$

where n_i is the number of sessions being executed at the criticality level i. This metric measures how many data units (specifically JPEG video frames) are saved from unnecessary dropping, given the possible worst-case dropping, $CLF_{\max}$, specified by the application users. In this experiment, we vary the total number of sessions submitted to the system from 50 to 400, in increments of 50.

Because the QoS maximization mechanism is independent of the criticality-based optimization dealt with by LP and PD, we simply pick up LP for the QoS performance evaluation. Figure 5-5 shows the performance of LP and QB against the AQI metric. The x axis values shown in the square brackets represent the number of executing sessions measured at run time. Of course, the AQI value under QB is zero, meaning no QoS improvement, as QB never readjusts the QoS of the scheduled sessions. With the QoS expansion operation under LP, the AQI value increases as the number of schedulable sessions increases. A saturation point is reached at 300 [99], beyond which the AQI start decreasing. This is because, as the degree of resource contention becomes higher, there is less room available for QoS expansion. Overall, the system resource manager performing the QoS expansion significantly improves the application performance with respect to AQI.

5.3. System Implementation

The QoS- and criticality-based multiresource scheduling approach has been implemented in the P*resto* system on a Sun SPARC20/Solaris 2.5 with C++, Parallax JPEG video card, SCSI disks, and video and audio devices (Huang, 1995). Recently, a "CPU only" version of the multiresource scheduler has been ported on a Pentium PC platform/Windows NT 4.0 with Visual C++ and QuickTime Video (Huang et al., 1997b).

To illustrate the basic implementation approach, Figure 5-6 shows the system software objects and the object interaction mechanism implemented along the path of system resource management (SRM) and CPU scheduling. Three classes of objects are defined: Session with each object instance representing an application, SRM (a single object instance), and CPUScheduler (a single object instance). The SRM and CPUScheduler provide operating-system-independent resource management services. The CPUScheduler services are implemented using Solaris' thread and Light Weight Process (LWP) manipulation services and Windows NT 4.0's thread manipulation services, respectively.

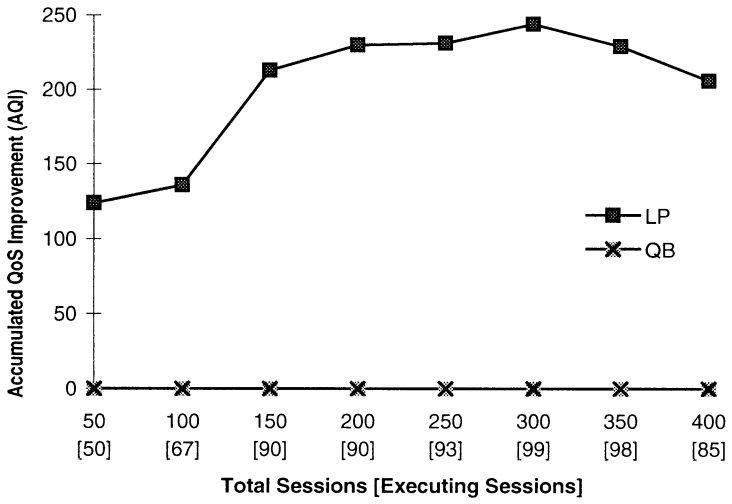

Figure 5-5. Effect of QoS expansion.

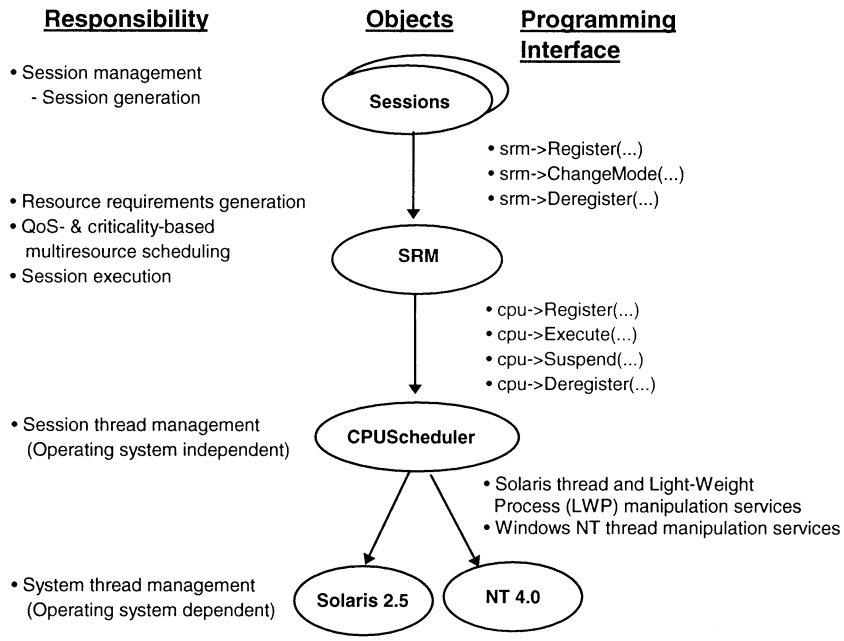

Figure 5-6. An instance of P*resto* software design.

A user interface has been developed to enable the application users to negotiate on line for available system resources by setting the value of criticality, QoS, or media flow rate for their applications. In addition to the system resource management capability, the P*resto* system provides a block-based application programming model associated with a visual programming tool and a continuous multimedia file system built on top of UNIX raw I/O. All these capabilities have been reported and demonstrated elsewhere (Huang, 1995; Huang et al., 1997b).

6. Related Work

In recent years, many system resource management and scheduling techniques have been developed to support continuous multimedia applications (Huang, 1995). Among them, a few address the issue of multiresource scheduling. D. Anderson proposed a metascheduling approach and formulated high-level admission control conditions with CPU, buffer space, and disk I/O resources (Anderson, 1993). However, low-level system activities such as thread scheduling and stream preemption were not addressed. K. Ramakrishnan et al. prototyped a multiresource management system for multimedia servers (Ramakrishnan et al., 1995). It supports not only media streams, but also aperiodic tasks and non-real-time tasks. On the other hand, the system admission control is static in the sense that it does not support resource negotiation. S. Chatterjee and J. Strosnider developed a heterogeneous resource management framework for providing timing guarantees to distributed multimedia applications (Chatterjee, 1995). The work focuses on classification and modeling of multimedia data flow and heterogeneous resources and high-level analysis of end-to-end media flow latency. The issue of multiresource allocation optimization was investigated by J. Huang and D.-Z. Du (1994). But their work did not consider application criticality and QoS requirements.

System support for application QoS has been an important topic for multimedia system researchers and developers (Vogel et al., 1995). H. Tokuda and T. Kitayama (1993) developed a QoS-based admission control technique in an end system that allows on-line resource negotiation in terms of spatial and temporal constraints of media data. Although their work did not deal with application criticality and multiresource optimization issues, it inspired our work on dynamic QoS negotiation. Dynamic QoS-based scheduling was also reported recently in Kaneko et al. (1996). The work mainly focused on the CPU resource and did not address how the task QoS is specified by applications.

Supporting application criticality has long been an issue of system resource management in the real-time community (Stankovic & Ramaritham, 1988); however, it was not addressed in the context of either multimedia or multiresource allocation optimization.

The uniqueness of our work lies in the fact that it considers application criticality as well as QoS and stream rate in multiresource scheduling and that it addresses optimization issues in the context of dynamic resource negotiation. It also provides a scheduling mechanism that enables application users to make on-line tradeoffs among application criticality, media QoS, and stream rate.

7. Conclusions

Presented in this paper is the design and performance analysis of a multiresource management system of the P*resto* multimedia environment developed at Honeywell for mission-critical multimedia applications. We introduced the notion of criticality to capture the semantics of application importance. We developed a multiresource admission control and scheduling approach that is able to support higher-criticality multimedia streams through the mechanisms of on-line QoS negotiation and multiresource preemption. To minimize the number of applications from preemption, we developed two approximation algorithms: one based on linear programming and the other on the primal-dual theorem. To maximize the number of executing applications and their QoS, we introduced a dynamic two-phase QoS adjustment approach.

Through analysis and experimentation, we compared the performance of our algorithms against both their upper (optimum) and lower (baseline) bounds. For the criticality-based multiresource preemption approach, we showed that the difference between the linear-programming-based approximation solution and the optimal solution is at most 4 (i.e., the approximation solution schedules at most 4 streams less than the optimal one does); that both the approximation algorithms (the linear-programming-based and the primal-dual-algorithm-based) perform better than the baseline scheme, which does random preemption; and that the primal-dual-algorithm-based approximation solution can reduce the computation overhead of the linear-programming-based solution by nearly a magnitude of two and is feasible for on-line scheduling use. We further showed that the QoS expansion approach can significantly increase the QoSs of executing video streams.

The multiresource management system has been implemented on a Sun SPARCstation 20/Solaris 2.5 and is being used in the P*resto* multimedia environment. A version of the scheduler with the capability of QoS-based CPU scheduling has been ported on a Pentium PC/NT4.0.

The work presented in this paper is being extended to provide end-to-end resource management services with the capabilities of distributed QoS negotiation and session preemption. The design and prototyping work is reported elsewhere (Huang et al., 1997a, 1997d).

Acknowledgments

The authors would like to thank M. Agrawal, D. Kenchamana-Hosekote, Jim Richardson, Satya Prabhakar, Duminda Wijesekera, and Eric Engtrom for their work on design and implementation of the P*resto* software infrastructure, multimedia file system, user interface, and block-based programming tool. Also thanks to the anonymous reviewers for their constructive comments on the early version of this paper.

Notes

1. Distributed resource management with network scheduling is out of the scope of this paper. We report our extended work on distributed scheduling elsewhere (Huang et al., 1997a, 1997d).

2. The term "basic feasible solution" is explained as follows. The linear programming considered here is to minimize a linear function over a region bounded by linear constraints. This region is called a feasible region. Every point in the feasible region is called a feasible solution. Since the boundary of the feasible region is formed by the linear constraints, the feasible region for the linear programming is a polyhedra. Moreover, each variable varies in [0,1]. Thus, the feasible region considered here is actually a polytope. (A polytopy is a bounded polyhedra.) Each vertex of this polytope is called a *basic feasible solution*. A fundamental result about the linear programming is that the optimal solution can be found among basic feasible solutions.

References

Anderson, D. 1993. Metascheduling for continuous media. *ACM Transactions on Computer Systems* 11: 3.

Chatterjee, S., and Strosnider, J. 1995. A generalized admission control strategy for heterogeneous, distributed multimedia systems. *Proceedings of the Third ACM International Multimedia Conference*, San Fransisco.

Fang, S.-C. and Puthenpura, S. 1996. *Linear Optimization and Extensions: Theory and Algorithm*. Prentice Hall.

Gemmell, D. J., Vin, H. M., Kandlur, D. D., Rangan, P. V., and Rowe, L. A. 1995. Multimedia storage servers: A tutorial. *IEEE Computer* May.

Guha, A., Pavan, A., Liu, J. C. L., and Roberts, B. A. 1995. Controlling the 97ess with distributed multimedia. *IEEE Multimedia* Summer.

Huang, J., and Du, D.-Z. 1994. Resource management for continuous multimedia database applications. *Proceedings of the 15th IEEE Real-Time Systems Symposium*, Puerto Rico, December.

Huang, J. 1995. Real-time scheduling technology for continuous multimedia applications. *Lecture Notes of the 3rd ACM Multimedia Conference*, San Francisco, October.

Huang, J., and Wan, P.-J. 1996. On Supporting Mission-Critical Multimedia Applications. *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*, June.

Huang, J., Wang, Y., Vaidyanathan, N. R., and Cao, F. 1997a. GRMS: A global resource management system for distributed QoS and criticality support. *Proceedings of the 4th IEEE Intl. Conference on Multimedia Computing and Systems*, June.

Huang, J., Heimerdinger, W., and Kappler, P. 1997b. QoS-Based Middleware Scheduling on NT. Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN 55418, August, Honeywell Technical Report SST-R97-004.

Huang, J., Kenchamanna-Hosekote, D., Agrawal, M., and Richardson, J. 1997c. **P***resto*—A system environment for mission-critical multimedia applications. *Journal of Real-Time Systems* September.

Huang, J., Jha, R., Heimerdinger, W., Muhammad, M., Lauzac, S., Kannikeswaran, B., Schwan, K., Zhao, W., and Bettati, R. 1997d. RT-ARM: A real-time adaptive resource management system for distributed mission-critical applications. *Proceedings of the IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December.

Jeffay, K. 1993. The real-time producer/consumer paradigm: A paradigm for the construction of effective, predictable real-time systems. *Proceedings of the 8th SIGAPP Symposium on Applied Computing*.

Kaneko, H., Stankovic, J. A., Sen, S., and Ramamritham, K. 1996. Integrated scheduling of multimedia and hard-real-time tasks. *Proceedings of the IEEE Real-Time Systems Symposium*, December.

Kenchammana-Hosekote, D., and Srivastava, J. 1996. I/O scheduling for digital continuous media. *ACM Multimedia Systems Journal*.

Liu, C. L., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *JACM* 20.

Liu, J. W. S., Lin, K.-J., Shih, W.-K., Yu, A. C., Chung, J.-Y., and Zhao, W. 1991. Algorithms for scheduling imprecise computations. *IEEE Computer*, May.

Niehaus, D. 1991. Program representation and translation for predictable real-time systems. *Proceedings of the 12th IEEE Real-Time Systems Symposium*, December.

Ramakrishnan, K. K., Vaitzblit, L., Gary, C., Vahalia, U., Ting, D., Tzelnic, P., Glaser, S., and Duso, W. 1995. Operating system support for a video-on-demand file service. *ACM Multimedia Systems* 3.

Robinson, S. R. (Ed.). 1993. *Emerging Systems and Technologies*. SPIE Optical Engineering Press.

Stankovic, J., and Ramamritham, K. (Eds.). 1988. *Tutorial: Hard-Real-Time Systems*. IEEE Computer Society Press.

SunSoft. 1995. Programming utilities guide: TNF utilities (beta release). SunSoft document (beta draft), 2550 Garcia Avenue, Mountain View, CA 94043.

Tokuda, H., and Kitayama, T. 1993. Dynamic QoS control based on real-time threads. *Proceedings of the 4th International Workshop on Network Support for Digital Audio and Video*, Lancaster, U.K., November.

Vin, H., Goyal, A., and Goyal, P. 1995. Algorithms for designing multimedia servers. *Computer Communications* 18(3): 192–203.

Vogel, A., Kerherve, B., von Bochmann, G., and Gecsei, J. 1995. Distributed multimedia and QoS: a survey. *IEEE Multimedia*, Summer.