On Supporting Mission-Critical Multimedia Applications*

J. Huang and P.-J. Wan
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418, USA
huang@htc.honeywell.com

Abstract

Mission-critical multimedia applications such as advanced command and control are dynamic and criticality-driven. Taking into consideration of the application criticality as well as media quality-of-service (QoS) and timing requirements, we develop a multiresource management system that enables criticality-based resource preemption and QoS-based dynamic resource negotiation and adaptation. Our performance study indicates that our approximation solution is near optimal and that it outperforms a criticality-cognitive baseline algorithm. We also show that the dynamic QoS adjustment technique largely improves the quality of service for video streams. The criticality- and QoS-based multiresource management system is part of the Presto multimedia system environment prototyped at Honeywell for mission-critical applications.

1. Introduction

Continuous multimedia, comprising video, audio, and image streams, is becoming an important element in the next generation of mission-critical applications such as crisis management and command and control. Unlike multimedia applications that are being developed for the entertainment (e.g., video-on-demand services) and office automation (e.g., video conferencing) industries, multimedia in mission-critical applications is unique in several ways. One characteristic is that media streams may be associated with an attribute of criticality—the importance of applications. For instance, an application performing periodic image capturing and flaw detection in advanced process control [3] 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 resources. In digital battlefield 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 [12]. 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.

Toward the goal of supporting mission-critical multimedia applications, we have developed and prototyped a resource management system that enables quality-of-service (QoS)-based dynamic resource negotiation and adaptation and criticality-based resource preemption [5]. We characterize the applications with three attributes: media stream flow rate, 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. Then the problem is how 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-complete) resource management problem consists of a number of 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

* This work was supported in part by Rome Laboratory under Contract F30602-93-C-0172 and by Honeywell Initiative R&D Program under Grant I4560-ME-2000.

0-8186-7436-9/96 $5.00 © 1996 IEEE
Proceedings of MULTIMEDIA '96

Authorized licensed use limited to: CityU. Downloaded on May 22,2010 at 14:41:19 UTC from IEEE Xplore. Restrictions apply.

concurrent streams. The second phase, called the *expansion phase*, expends 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. Using an approximation algorithm, the system preempts low-criticality stream(s) and readjusts the QoS of executing streams toward the goal of supporting high-criticality applications as well as providing the best QoS service. Finally, as a result of QoS adjustment and criticality-based preemption, an on-line resource negotiation and adaptation mechanism is provided. 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 performance analysis shows that the designed resource management system performs much better than a criticality-cognitive baseline approach and that the difference between our approximation solution and the optimal solution is at most 4. We also show that the dynamic QoS expansion technique can significantly improve the quality of service for video streams. Currently, the system is running in a multimedia system environment, called P*resto*, comprising Sun SPARCstation 20, the Solaris 2.4 operating system, Parallax JPEG video, a multimedia file system, and user interface and a block-based application programming tool for command and control applications.

The problem of scheduling multiple system resources for concurrent media streams has been investigated recently [1, 4, 10]. The work presented in this paper uniquely incorporates the applications' criticality as well as QoS properties into the scheduling framework and enables applications to make on-line tradeoffs between their criticality, QoS, and rate specifications.

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 the system resource management architecture and associated scheduling algorithms for the individual resources. The core of this paper, namely dynamic QoS-based multiresource negotiation and adaptation and criticality-based resource preemption, is presented in Section 4. In Section 5, we discuss the related work. Finally, we conclude this paper and point out future work in Section 6.

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 when 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, etc., which depend largely on application semantics. As a starting point, we define the QoS as Consecutive Loss Factor (CLF) in this work. CLF is the maximum number of consecutive data units allowed to be dropped between every two processed units. In particular, the application specifies its CLF using a range $[0, CLF_{max}]$. At run time, the application may adapt its CLF between 0 and CLF_{max} depending on the availability of system resources.
- *Criticality*—Criticality refers to the degree of application importance among concurrent applications. Application criticality is classified by multiple levels. In the case of resource contention, applications with higher criticality shall be allocated resources first.

Note that the timing constraints, QoS, and criticality are orthogonal to each other; i.e., a user may specify any of the parameter values independent 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.

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.

Authorized licensed use limited to: CityU. Downloaded on May 22,2010 at 14:41:19 UTC from IEEE Xplore. Restrictions apply.

3.1. Session Model

We use the notion of *session* to capture the execution behavior of continuous media applications. Following the real-time producer-consumer paradigm [9, 4], a session consists of producer and consumer threads and a doublebuffer between the producer and the consumer. 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 a unit of resource allocation and scheduling.

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 by the underlying scheduling algorithm. Therefore, the actual stream flow rate of the session will be

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

- 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.

- m_i—is the double buffer allocated to a session. Its size is equal to ($2x_iu_i$), 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. Currently, these services are provided through the POSIX standard operating system interface. 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 that manage their own resources on the basis of threads, I/O processes, buffer requests, and video processing and display, respectively. In particular, they carry out scheduling operations. Their scheduling algorithms are actually exercised by the system resource manager for systemwide resource management.

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 high-lighted in the figure, this paper focuses on the system resource manager, discussing its criticality-and QoS-based multiresource scheduling.

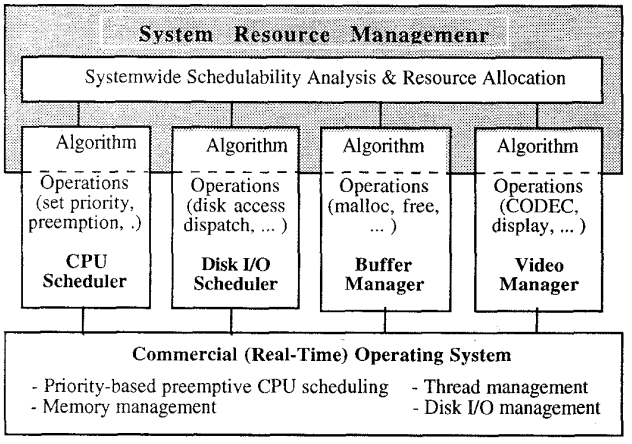

Figure 3-1. System Resource Management Infrastructure

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.

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 a simple interval-based I/O access policy [2]. Let

$$L = \min(L_i), \quad where \quad 1 \le i \le n$$

That is, L is the latency tolerable by all the n applications. We use L as the time interval for scheduling of concurrent media streams. Assume that the amount of contiguous data that the disk can transfer in one second is $Dmax$ and the average disk seek time for serving each I/O request within L is S. Then, during the time interval L,

Authorized licensed use limited to: CityU. Downloaded on May 22,2010 at 14:41:19 UTC from IEEE Xplore. Restrictions apply.

the effective transfer time is $L - nS$. Therefore, the n sessions can be schedulable only if

$$\sum_{i=1}^{n} x_i u_i \le (L - nS) D_{\max}$$

where $x_i = \lceil r_i L \rceil$, which is the number of data units processed within L. Or equivalently,

$$\sum_{i=1}^{n} x_i u_i + nSD_{\max} \le LD_{\max} \quad (1)$$

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

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

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

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 $\sum_{i=1}^{n} r_i$. Let $V_{\max}$ be the maximum supportable video rate. Then n sessions can be schedulable if

$$\sum_{i=1}^{n} r_i \le V_{\max} \quad (3)$$

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. If the maximum memory space available is M_{max} bytes, n sessions can be supported if

$$2\sum_{i=1}^{n} x_i u_i \le M_{\max} \quad (4)$$

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

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

4.1. Approach

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 re-negotiate on line, called "interactive QoS negotiation", with the application for its willingness to lower its QoS specification.

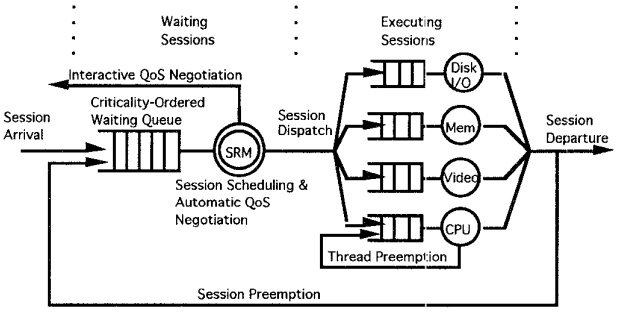

Figure 4-1. Scheduling Mechanism

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 finished 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 the executing sessions to their minimums (i.e., $CLFa = CLF_{max}$) to yield the resources to waiting sessions. Its objective is to

Authorized licensed use limited to: CityU. Downloaded on May 22,2010 at 14:41:19 UTC from IEEE Xplore. Restrictions apply.

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., $CLFa = 0$). 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

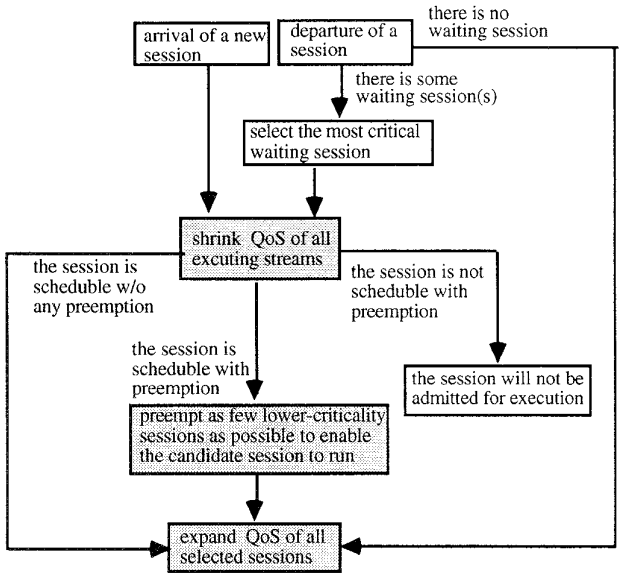

Figure 4-2. The Scheduling Strategy

higher-criticality session is not schedulable. The goal is to serve the higher-criticality session and at the same time to preempt as fewer sessions as possible.

In the following, we discuss the design of the algorithms in detail.

QoS Shrinking—Each time when scheduling a candidate session in the waiting queue, we first reduce the QoSs of all executing sessions to their lowest level, i.e., $CLFa_i = CLFmax_i$. Then we check if the new session is executable without any preemption. If so, we expand the QoS of all the executing sessions in the QoS expansion phase. If not, we consider session preemption.

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 one level as the *schedulable criticality level*, above which all the sessions are schedulable after insertion of the candidate session. The executing sessions, some in the schedulable criticality level and some below the schedulable level, must be preempted. For the preemption process, we first need to find the schedulable criticality level and then add back as many sessions as possible from the levels at or below the schedulable criticality level.

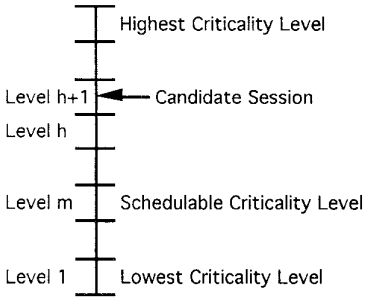

Figure 4-3. Criticality Levels

Find the Schedulable Criticality Level—We use a binary search procedure to find the schedulable criticality level, m, among the h levels.

Add Back—We try to add back as many sessions as possible from the m levels. This process is performed level by level from the criticality level m to the lowest criticality level 1. At each level, we assign priorities to individual sessions such that the session with the highest priority is added back first whenever the resources are available. Since finding the optimal priority assignment is NP-hard, we consider a suboptimal priority assignment algorithm. Let the remaining CPU, video processor, memory, and disk I/O resources available to sessions at or below level k be denoted by Crem, Vrem, Mrem, and Drem. 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 variable y_{kj} and solve the following linear programming.

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

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

$$\sum_{j=1}^{n_k} y_{kj}x_{kj} \le V_{rem}$$

$$\sum_{j=1}^{n_k} y_{kj}M_{kj} \le M_{rem}$$

$$\sum_{j=1}^{n_k} y_{kj}(x_{kj}u_{kj} + SD_{\max}) \le D_{rem}$$

$$0 \le y_{kj} \le 1, \quad 1 \le j \le n_k$$

Authorized licensed use limited to: CityU. Downloaded on May 22,2010 at 14:41:19 UTC from IEEE Xplore. Restrictions apply.

Here we propose an approximation algorithm for the priority assignment process.

Approximation Algorithm

Step 1. Solve (SLP_k). *Let* $(y_{kj}, j = 1, \ldots, n_k)$ *be an optimal solution.*

Step 2. Order y_{kj} *such that*

$$y_{k1'} \geq y_{k2'} \geq \ldots \geq y_{kn_k'}$$

Step 3. Assign session priorities according to the order obtained from Step 2.

After finishing the priority assignment, the adding back process can be performed according to the order of priorities.

QoS Expansion—Now we expand the QoS of all the schedulable sessions. 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 CLFmax is Q and the maximum number of sessions is N, then the QoS expansion procedure will take time $O(QN)$.

4.2. Performance Analysis

4.2.1. Optimality Analysis

The following theorem states the suboptimality of our approximation algorithm used in the criticality-based session preemption process.

Theorem. The difference between the number of sessions added back under our priority assignment and the maximum number of sessions that could be added is at most four.

For proof of this theorem, the reader is referred to [7].

4.2.2. Performance Evaluation

We have conducted performance evaluation of the scheduling algorithms through simulation. For comparison, we consider three system resource schedulers in terms of their criticality-preemption and QoS-adjustment schemes. The first scheduler is our system resource manager (SRM) that employs an approximation optimization algorithm for criticality-based preemption and performs both QoS shrinking and expansion for QoS negotiation. The second scheduler is a criticality baseline scheduler (CB) in the sense that like SRM, it performs session preemption in the order of increasing criticality levels. But different from SRM, it does not consider optimization while performing preemption within a criticality level. The third scheduler is a QoS baseline scheduler (QB) which uses the same approximation algorithm as SRM for preemption optimization, but it does not perform QoS expansion. Our objective is to understand the effect of our optimized preemption scheme by comparing SRM against CB, and the effect of our QoS expansion technique by comparing SRM against QB.

(For detailed information about the simulation workload and system parameter settings, the reader is referred to [7].)

Effect of Criticality-Based Preemption—As shown in Table 4-1, 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 system resource manager (SRM) and the baseline approach (CB) in terms of the number of sessions actually being executed. As indicated in the table, SRM and CB are comparable when the number of sessions in the system is set at 50. This is because there 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 the level 1 are completely preempted at the level 1 under CB. When there are more sessions in the system (300), only higher-criticality sessions can be executed. Now compare SRM and CB at the criticality level 4: SRM scheduled 42 session, while CB scheduled 23. Clearly, our approximate optimization algorithm performs significantly better than the baseline approach which is criticality-cognitive, but does random preemption within each criticality level.

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 Subsection 4.1. In particular, we compare SRM with QB—a baseline approach which does not increase QoS of the scheduled sessions even if there are some "left-over" resources. We

Authorized licensed use limited to: CityU. Downloaded on May 22,2010 at 14:41:19 UTC from IEEE Xplore. Restrictions apply.

Table 4-1. Effect of criticality-based session preemption

Criticality	Total Number of Sessions vs. Number of Scheduled Sessions											
	50		100		150		200		250		300	
	SRM	CB	SRM	CB	SRM	CB	SRM	CB	SRM	CB	SRM	CB
Level 5 (High)	6	6	17	17	26	26	36	36	47	47	57	57
Level 4	14	14	24	24	38	38	46	38	46	30	42	23
Level 3	13	13	22	22	24	11	8	0	0	0	0	0
Level 2	11	11	3	2	0	0	0	0	0	0	0	0
Level 1 (Low)	6	6	1	0	0	0	0	0	0	0	0	0

define a performance metric, called *Accumulated QoS Improvement* (*AQI*), as

$$AQI = \sum_{i=1}^{Cmax} \sum_{j=1}^{n_i} (CLF\max_{ij} - CLFa_{ij})$$

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

Figure 4-4 shows the performance of SRM 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, since QB never readjusts the QoS of the scheduled sessions. Under SRM, 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, SRM significantly improves the application performance with respect to AQI.

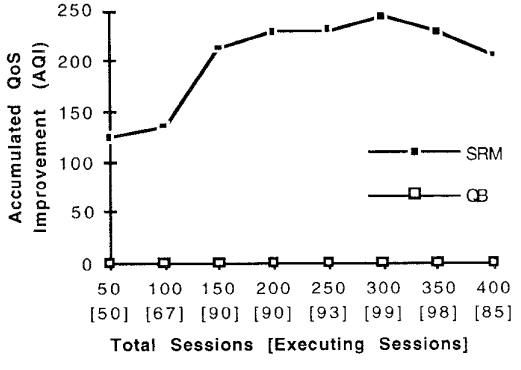

Figure 4-4. Effect of QoS Expansion

5. Related Work

In recent years, many system resource management and scheduling techniques have been developed to support continuous multimedia applications [6]. 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 [1]. However, the 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 [10]. 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. The issue of multiresource allocation optimization was investigated by J. Huang and D.-Z. Du [4]. But they did not consider application QoS and criticality requirements.

System support for application QoS has been an important topic for multimedia system researchers and developers [15]. H. Tokuda and T. Kitayama 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 [14]. Although their work did not deal with application criticality and multiresource optimization issues, it inspired our work on dynamic QoS negotiation.

Supporting application criticality has long been an issue of system resource management in the real-time community [13]. However, it was not addressed in the context of either multimedia or multiresource allocation optimization.

Authorized licensed use limited to: CityU. Downloaded on May 22,2010 at 14:41:19 UTC from IEEE Xplore. Restrictions apply.

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 on line make tradeoffs between application criticality and QoS.

6. Concluding Remarks

Presented in this paper is the design, implementation, and performance analysis of a multiresource scheduler of the *Presto* multimedia system developed at Honeywell for mission-critical multimedia applications. We introduced the notion of application criticality to capture the semantics of application importance. We developed a multiresource scheduling scheme that is able to support higher-criticality applications through the mechanisms of on-line QoS negotiation and session preemption. As part of the scheduling scheme, a dynamic two-phase QoS adjustment technique was developed to maximize the application QoS and the number of executing applications.

Through theoretical analysis and simulation, we compared the performance of our algorithm against both its upper bound (optimum) and lower bound (baseline). For the criticality-based preemption technique, we showed that the difference between our approximation solution and the optimal solution is at most 4, and that the approximation algorithm performs significantly better than the baseline scheme which does random preemption (no optimization) within a criticality level. We further showed that the QoS expansion technique can significantly increase the QoSs of executing video streams. An early version of the multiresource scheduler has been implemented and being used in the *Presto* system.

This work is being extended in several ways. We are currently extending the central multiresource scheduler to a set of distributed, decentralized end-to-end scheduling components for distributed multimedia applications [8]. Our future work will extend the dynamic QoS adjustment technique to handle MPEG-like media streams, which must take into account the relationship between frames.

Acknowledgment

The authors would like to thank M. Agrawal, D. Kenchamana-Hosekote, Jim Richardson, Satya Prabhakar, and Eric Engtrom for their work on design and implementation of *Presto* software infrastructure, multimedia file system, user interface, and block-based programming tool.

References

[1] D. Anderson, "Metascheduling for Continuous Media," *ACM Transactions on Computer Systems*, Vol. 11, No. 3, August 1993.

[2] J. Gemmell, et al. "Multimedia Storage Servers: A Tutorial," *IEEE Computer,* May 1995.

[3] A. Guha, et al., "Controlling the Process with Distributed Multimedia," *IEEE Multimedia*, Summer 1995.

[4] J. Huang, and D.-Z. Du, "Resource Management for Continuous Multimedia Database Applications," *Proceedings of the 15th IEEE Real-Time Systems Symposium*, Puerto Rico, December 1994.

[5] J. Huang, "A System Software that On-line Supports Criticality and QoS of Continuous Multimedia Streams," *Honeywell Patent Disclosure H16 16120*, February 1995.

[6] J. Huang, "Real-Time Scheduling Technology for Continuous Multimedia Applications," *Lecture Notes of the 3rd ACM Multimedia Conference*, San Francisco, November 1995.

[7] J. Huang and P. Wan, "On Supporting Mission-Critical Multimedia Applications," Honeywell Technical Report SST-R95-011, November 1995.

[8] J. Huang, Y. Wang, and D. Kenchammana-Hosekote, "A Decentralized End-to-End Scheduling Approach for Continuous Multimedia Applications," *Proceedings of the 6th International Workshop on Network and Operating System Support for Digital Audio and Video*, Japan, April 1996.

[9] K. Jeffay, "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*, 1993.

[10] K.K. Ramakrishnan, et al., "Operating System Support for a Video-on-Demand File Service," *ACM Multimedia Systems (3)*, 1995.

[11] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, 20, January 1973.

[12] S.R. Robinson, Editor, "Emerging Systems and Technologies," *SPIE Optical Engineering Press*, 1993.

[13] J. Stankovic and K. Ramaritham, Editors, "Tutorial: Hard Real-Time Systems," *IEEE Computer Society Press*, 1988.

[14] H. Tokuda and T. Kitayama, "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 1993.

[15] A. Vogel, et. al., "Distributed Multimedia and QoS: A Survey," *IEEE Multimedia*, Summer 1995.

Authorized licensed use limited to: CityU. Downloaded on May 22,2010 at 14:41:19 UTC from IEEE Xplore. Restrictions apply.