QOS-Ticket: A New Resource-Management Mechanism for Dynamic QOS Control of Multimedia*

Kiyokuni KAWACHIYA

IBM Research,
Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato,
Kanagawa 242, Japan

Hideyuki TOKUDA

Faculty of Environmental Information,
Keio University
5322 Endo, Fujisawa,
Kanagawa 252, Japan

Abstract

In an interactive multimedia environment that handles multiple sessions dynamically, a mechanism for controlling the QOS among sessions is very important. It is, however, difficult to achieve such QOS control without any system support. A new resource-management mechanism and QOS-control architecture for multimedia applications are needed. This paper first discusses features required for such architecture, and then proposes a new QOS-control architecture based on "QOS-Ticket." In this QOS-Ticket model, resource management and QOS control are achieved through the cooperation of an operating system, a QOS Manager, and individual continuous-media sessions. The QOS-Ticket for each session, which represents that session's resource reservation, mediates these activities. A prototype of the QOS-Ticket model has been implemented on Real-Time Mach, using the system's processor capacity reservation mechanism. An experiment with this prototype is described, to show the effectiveness of the QOS-Ticket model.

Keywords: Dynamic QOS control, Resource management, Real-time operating system, Multimedia

1 Introduction

The Keio-MMP (MultiMedia Platform) project has as its aim the development of a common software platform for distributed multimedia computing [1]. One goal of the project is to incorporate the concept of quality of service (QOS), and to manage system resources over multiple continuous-media sessions on the basis of this concept. However, it is difficult for multimedia applications to manage their QOS appropriately by themselves without system support. We therefore selected Real-Time Mach 3.0 (RT-Mach) as a basis for such system support. We are extending RT-Mach and constructing servers for multimedia processing on our extension [2, 3, 4].

*This research is conducted under the Open Fundamental Software Technology Project of Information-technology Promotion Agency, Japan (IPA).

In an interactive multimedia environment that handles multiple sessions dynamically, a mechanism for controlling QOS among sessions is very important. This mechanism is closely related to the system's resource management. However, most existing resource-management mechanisms focus on the "fairness" of the resource assignment, and provide insufficient support for the resource guarantee, enforcement, and observation. Therefore, a new resource-management mechanism should be considered as a means of providing appropriate QOS control [5].

In accordance with these requirements, this paper proposes a new QOS-control model based on "QOS-Ticket," and describes the resource management needed to implement the model. A prototype of the QOS-Ticket model has been implemented on RT-Mach, using the system's processor capacity reservation mechanism. An experiment with this prototype is described, to show the effectiveness of the QOS-Ticket model. In the rest of this paper, Section 2 discusses features required for the QOS-control architecture. Section 3 proposes a QOS-Ticket model that satisfies all the requirements. Section 4 describes a prototype implementation of the QOS-Ticket model, named QOS-Control Server, and Section 5 describes an experiment using the prototype. Sections 6 and 7 discuss future and related work on QOS control of multimedia, and Section 8 offers some conclusions.

2 Features Required for a QOS-Control Model

The most notable characteristic of data for continuous media such as audio and video is the existence of constraints on the timing of the data processing. To handle such data appropriately, the system must observe the timing constraints, and must guarantee sufficient system resources (CPU, memory, disk, network, etc.) for the processing. Of course, since the real system resources are finite, sufficient resources may not be provided for a particular processing session. In such cases, the execution time is usually extended. But in continuous-media processing with timing constraints, the QOS should be changed to overcome the resource shortage [6]. For example, in video processing, the

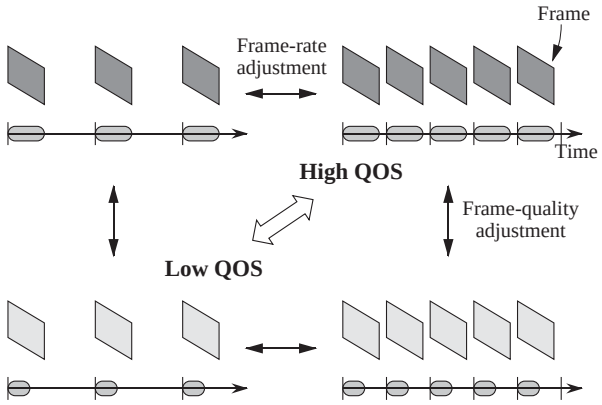

Figure 1: Example of QOS management: video processing

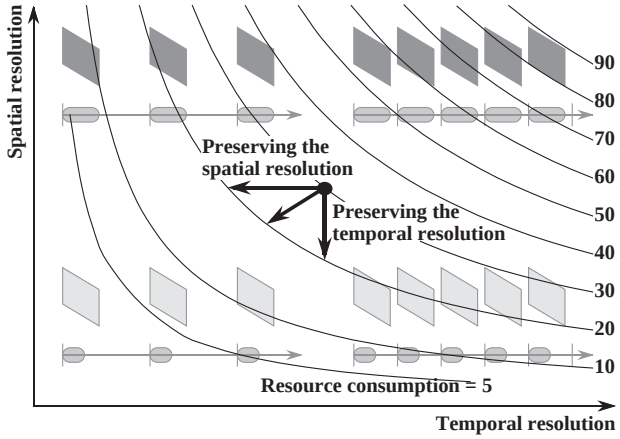

Figure 2: Various ways of changing the QOS

frame rate and/or the quality of each frame can be changed to meet the resource restriction. Figure 1 illustrates an example of this QOS management. The upper-right section represents high-QOS processing, and the lower-left section represents low-QOS processing, which requires few resources.

To support multiple continuous-media sessions dynamically in a system, some framework for resource management and QOS control is necessary. The following features are considered to be required for such a framework.

1. **Resource reservation and restriction**

   The usual "fair-share" resource management is insufficient to allow each session to meet its timing constraints. A mechanism for guaranteeing (reserving) the resource assignment is necessary. The mechanism must also have a resource-restriction function to prevent bad-mannered sessions from exhausting resources. Moreover, this resource management should be applicable to each session rather than to each process or thread, because a session is usually made up of multiple processes or threads.

2. **Resource-use adaptation**

   The QOS-control framework should not depend on some specific system configuration or set of media data. However, the resources needed for a continuous-media processing operation may vary according to the type of CPU, media data, or other factors. Therefore, it is difficult to know all the resource requirements in advance. It is thus desirable that some feedback and adaptation mechanism based on the actual resource consumption should be provided in the framework.

3. **Coordination of multiple sessions**

   When a system handles multiple sessions, the QOS-control mechanism should take account of the characteristics of the sessions. For example, if there is a resource shortage, the QOS of low-priority sessions should be reduced more than that of high-priority sessions.

4. **Separation of policy and mechanism**

   There are various ways of changing a session's QOS to meet a specific resource restriction. The graph in Fig. 2 illustrates the relations between the temporal and spatial resolutions for specific levels of resource consumption. Arrows in the figure show various ways of reducing the QOS when the available resources are decreased from 30 to 20. The policy for choosing one of these ways should be kept separate from the mechanism for QOS control. This separation of policy and mechanism is suitable for the microkernel environment [7].

We are now designing a new QOS-control model that satisfies all the above requirements [8].

3 QOS-Ticket Model

Figure 3 shows an outline of this model, which combines resource reservation and adaptation, whereas most previous mechanisms take account of only one of them. Its core part is a mechanism named "QOS-Ticket," which is used for resource reservation and restriction, and for monitoring. In this QOS-Ticket model, resource management and QOS control are achieved through the cooperation of an operating system, a QOS Manager, and individual continuous-media sessions. The QOS-Ticket for each session, which represents that session's resource reservation, mediates these activities.

In the QOS-Ticket model, as a rule, all the system resources are used with reservation. The model's basic behavior is as follows. Each continuous-media session registers its "QOS factor," which describes the characteristics of the session, with the QOS Manager at the time it is started. When a session is registered, the QOS Manager calculates the resource allocation

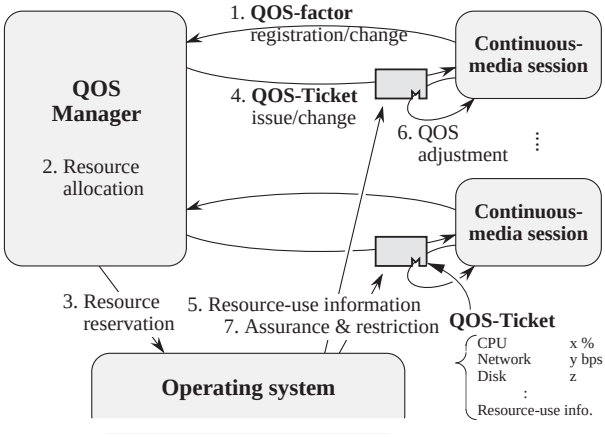

Figure 3: QOS-Ticket model

among sessions on the basis of the QOS factors, reserves resources, and issues a QOS-Ticket that contains the reservation information for the session. The session adjusts its own QOS to meet the resource restriction specified in the QOS-Ticket. The resource management needed for the QOS-Ticket is performed by the operating system.

The roles of the five components in the QOS-Ticket model are as follows:

QOS factor. The QOS factor, registered by each session with the QOS Manager, describes the characteristics of the session. It specifies a priority and a tolerable allocation range (maximum and minimum values) for each resource request. The QOS Manager will allocate resources to meet this request. The QOS factor may be modified (re-registered) if the priority or required resource is changed.

QOS-Ticket. The QOS-Ticket is a "ticket," issued by the QOS Manager to each session, for preferential use of resources. The QOS-Ticket represents reservation of resources for a session. Through the QOS-Ticket, a session can use resources preferentially, within the reserved limit. The resource consumption is cumulatively registered in the QOS-Ticket. When the consumption exceeds the reserved limit, the session can no longer use the resource preferentially. The QOS-Ticket should be a resource-management mechanism per session. Even if a session consists of multiple processes or threads, the QOS-Ticket is shared among them. When a session calls some server program, the QOS-Ticket is temporarily passed to the server via the RPC, and processing in the server can use resources preferentially. Through the QOS-Ticket, each session can find out the quantity of resources consumed, as well as its own resource-reservation status.

Operating system. The operating system provides a resource management mechanism for QOS-Ticket, consisting of the following items,

- Resource reservation and restriction
- Resource-use information

The resource reservation is indispensable for realizing QOS-Ticket. The resource restriction prevents bad-mannered sessions from disturbing other sessions. The resource-use information provides each session with important hints on how to control the QOS. Moreover, it is desirable that this resource control should be applicable to a "group" of threads that make up a session.

QOS Manager. The QOS Manager is a kind of scheduler that allocates resources to sessions. Basically, the allocation is calculated according to the QOS factors, but the policy is decided by the QOS Manager. On the basis of the allocation, the QOS Manager reserves resources and issues a QOS-Ticket to each session. When the number of sessions or some QOS factor is changed, the QOS Manager recalculates the resource allocation, modifies the resource reservation of the QOS-Ticket, and notifies each session of the change via upcall. In the case of a severe resource shortage, the QOS Manager may request that some low-priority sessions be suspended.

Continuous-media session. Each continuous-media session estimates the amount of resources needed for processing, and declares it to the QOS Manager as part of the QOS factor. The estimate can be revised during the processing according to the actual resource consumption. The requested resources are allocated by the QOS Manager, and a QOS-Ticket is issued. The session adjusts its own QOS to meet the resource restriction specified in the QOS-Ticket. The resource-use information in the QOS-Ticket can be used as a hint on the adjustment.[1] How to change the QOS is left up to the session; it may, for example, use the characteristics of the media or some QOS-control policy.

In the QOS-Ticket model, the requirements listed in Section 2 are satisfied as follows. The resource reservation and restriction are done by the operating system. The QOS-Ticket provides the resource-use information, which enables the feedback and adaptation mechanism to realize the independency of the system or media data. The QOS Manager takes charge of the coordination of multiple sessions.

The separation of policy and mechanism is achieved on two levels, as shown in Fig. 4. One level is that of resource allocation. The resource reservation mechanism resides in the operating system, while the resource allocation policy is up to the QOS Manager. The other level is that of QOS control. The resource allocation mechanism for QOS control is implemented in the QOS Manager (and operating system), but the QOS control based on the allocated resources is handled by each session according to its own policy. The

[1] For example, in the case of video processing, a session can record the amount of resources used by the previous frames, and can change the frame rate according to that information.

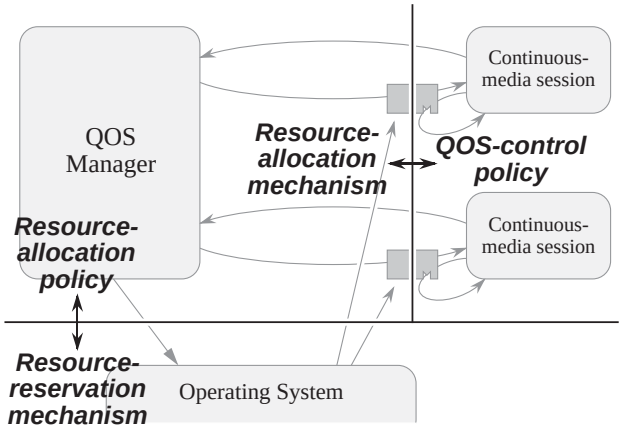

Figure 4: Separation of policy and mechanism in the QOS-Ticket model

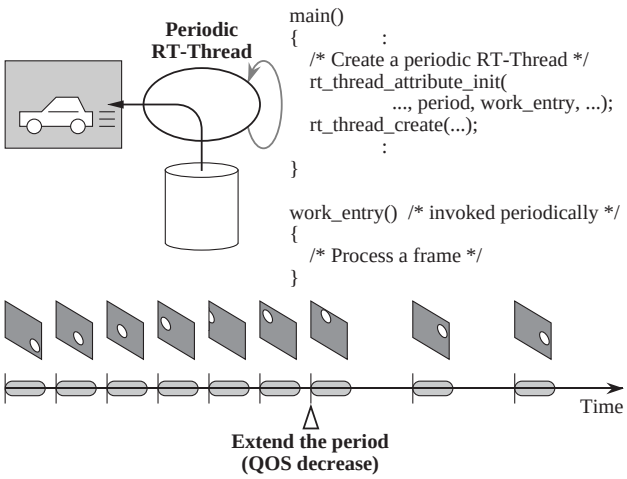

Figure 5: Continuous-media processing in RT-Mach

QOS-Ticket (and QOS factor) mediate these three activities. In general, the QOS Manager and the operating system have more knowledge on the resource usage, while each session knows more about the characteristics of the media. Therefore, we consider this separation to be very reasonable.

4 A QOS-Control Server Based on the QOS-Ticket Model

We have been implementing the QOS-Ticket model on RT-Mach, an operating system based on the Mach microkernel [9, 10] with several real-time extensions such as real-time threads, a real-time scheduler, real-time synchronization, and real-time inter-process communication (IPC) [11, 12, 13]. As shown in Fig. 5, continuous-media processing programs can be written in RT-Mach by using the system's periodic real-time thread [14], which processes a media data unit (MDU), such as a frame of video data, on every invocation. In this paper, such threads will be called "CM-Threads." One easy way of achieving QOS control is to change the period attribute of the CM-Thread, as shown in the lower part of Fig. 5 [15]. This corresponds to a change between left and right in Fig. 1.

The first prototype of the QOS-Ticket model is implemented on RT-Mach. This experimental QOS Manager, named QOS-Control Server, controls the CPU resource allocation among multiple CM-Threads. As the QOS-Ticket for the CPU resource, the "processor capacity reservation" mechanism [16, 17] of RT-Mach is used. This mechanism makes available in the system a new abstraction named "reserve." A reserve represents a CPU-resource reservation in a form of "requires C seconds every T seconds (i.e. $\frac{C}{T}$ of the CPU)." As shown in Fig. 6, every thread in the system is associated with a reserve, and can gain preferential use of the CPU through it. A reserve accumulates information on the CPU time used by its associated threads. On the basis of this information, in assigning the CPU, the scheduler gives top priority to threads whose reserves are not used up. If a reserve is exhausted, threads associated with it are executed in time-sharing mode until the next replenishing time only if there is no runnable thread that has an unexhausted reserve. Each thread can also access the CPU-time information in its reserve, and thus find out how much CPU time has been spent on its processing. It is also possible to associate multiple threads with one reserve or to pass a reserve to a server via RT-Mach IPC. For example, in Fig. 6, threads 2 and 3 share the same reserve B, and thread 4 is temporarily associated with reserve B while it is called from thread 3. In a word, the reserve can satisfy the features of the QOS-Ticket described in Section 3.

Figure 7 shows the structure of the QOS-Control Server using the processor capacity reservation. Each CM-Thread registers its QOS factor with the server via Mach IPC at the time it is started. In the current implementation, the range of periods that a CM-Thread can accept, the relative workload, and the priority (a high value means a high priority) of the thread are used as the attributes of a QOS factor. In accordance with the QOS factors, the server calculates the CPU-resource assignment for each session, and "issues" a reserve to each CM-Thread.[2] As described in Section 3, many policies can be considered for this assignment. In the current implementation, the CPU-resource assignment ($\frac{C_i}{T_i}$) to these reserves is calculated by the server according to the value of (workload $\times$ priority).

Each CM-Thread adjusts its own QOS (e.g., by adjusting its period) to meet the restriction imposed by its reserve. The CPU-time information in the reserve can be used as a hint in making this QOS adjustment. For example, through the reserve, a thread can know the CPU-resource assignment ($\frac{C_i}{T_i}$) and the CPU time consumed in each processing unit (c_i). In this case,

[2] Strictly speaking, the server creates a reserve for each CM-Thread, associates the thread with it, and signals the CM-Thread via upcall.

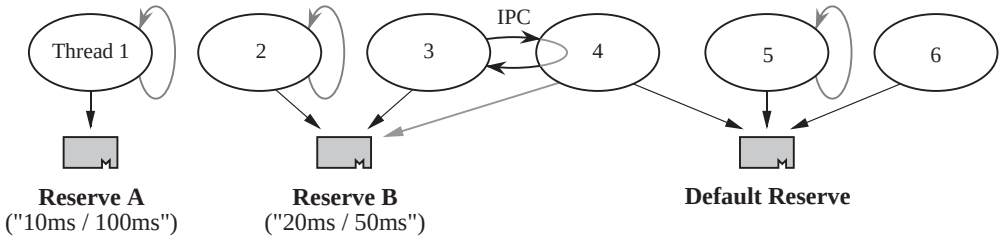

Figure 6: Overview of the processor capacity reserves

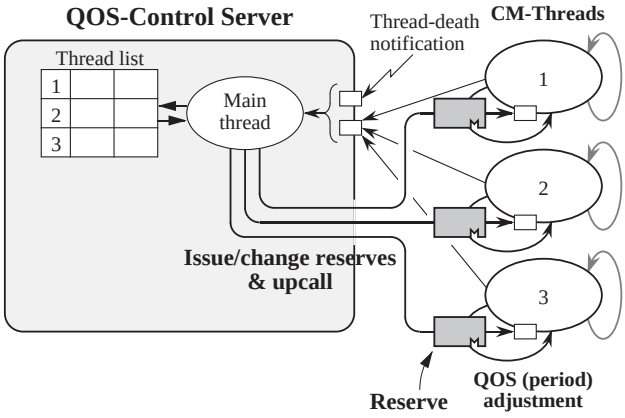

Figure 7: Structure of the QOS-Control Server

the period can be adjusted to $(c_i \times \frac{T_i}{C_i})$. We are now developing a user-level library to support such adjustment.

If a session is newly registered or finished, or the QOS factor of some session is changed, the server recalculates the CPU-resource assignment, sets it to the reserves, and notifies CM-Threads of the change via upcalls.

5 An Experiment with the QOS-Control Server

An experiment was carried out to determine how well the server (and the QOS-Ticket model) controls the QOS of multiple sessions. The four dummy CM-Threads shown in Table 1 were used in this experiment. These dummy threads were real-time threads that were periodically invoked to execute a specific number of dummy loops according to their workload values. Dummy thread 1 represents a session in which the work is light but the QOS cannot be changed, such as some types of audio-processing session. Dummy thread 2 allows a change in the QOS, but in the worst case requests a 100 msec period. Dummy threads 3 and 4 are sessions started during thread 2, and their priorities are higher than that of thread 2. These dummy threads change their own period to meet the restricted availability of the CPU resource imposed by their reserve, issued by the QOS-Control Server.

Table 1: QOS factors of dummy CM-Threads

No.	Exec. term (sec)	Work-load	Period (msec)	Pri-ority
1	$00 \rightarrow 60$	10	50 (Fixed)	100
2	$00 \rightarrow 60$	50	$30 - 100$	10
3	$10 \rightarrow 40$	50	$30 - \infty$	20
4	$20 \rightarrow 50$	50	$30 - \infty$	30
D	$30 \rightarrow 35$	(Disturbing thread that exhausts the CPU by looping infinitely)		

In addition to these dummy CM-Threads, a "disturbing" thread runs from 30 to 35 sec. This thread tries to exhaust the CPU by looping infinitely.

The experiment was carried out on an IBM ThinkPad 755C (9545-L, IntelDX4-75MHz) with RT-Mach MK94.[3] The resolution of the system clock was changed to 1 msec. Figure 8 shows the results. The upper part of this figure shows the actual period (QOS) of each dummy thread,[4] and the lower part shows the CPU utilization and deadline misses during the experiment.

In the first 10 seconds, the system load was light and threads 1 and 2 were able to run with their highest QOS (i.e. with the minimum period). But after threads 3 and 4 were started, the CPU assignment was recalculated and reduced by the QOS-Control Server according to the QOS factors, and the period of each thread was extended. Even during the 30 to 35 sec that the disturbing thread was running, the four dummy threads were able to run without any influence. When threads 3 and 4 were ended, the QOS of thread 2 was increased again. For thread 1, a fixed QOS (a 50 msec period) was achieved for the whole experiment.

In this experiment, the behavior of CM-Threads was very stable. Moreover, as the lower graph shows, CPU utilization was also stable at 80–90% (except while the disturbing thread was running), and the

[3] RT-Mach MK94 is a version released in 1994 by the Keio-MMP project, based on CMU's MK83i with some extensions.

[4] In the experiment, the period was used as a QOS value. Therefore, in the graph, the lower value represents the better QOS.

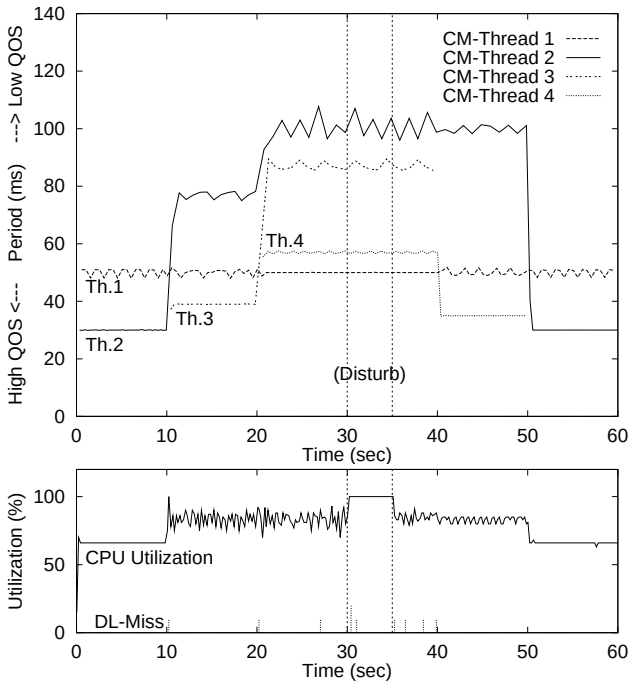

Figure 8: Actual result of QOS control (Exp. 3)

number of deadline misses was almost zero. In addition to having this stable behavior, the server is tolerant of disturbing ("bad-mannered") threads. If a thread tries to exhaust the CPU, it is prevented from doing so by the reservation mechanism.

6 Current and Future Work

The experiment shows that the QOS of the CPU resource can be effectively controlled by the QOS-Control Server in accordance with the QOS-Ticket model. Our ultimate target is to manage all system resources — not only the CPU, but also the memory, disk, network, and so on — in an integrated fashion within the framework of the QOS-Ticket model. Currently, the resource-control mechanisms for QOS-Ticket are being added to RT-Mach. As shown in the experiment, the processor capacity reservation mechanism can be used for managing the CPU resource. For managing network resources, reservation-based protocols such as ST-II [18] and RSVP [19] can be used. In our project, an ST-II protocol server is being developed on RT-Mach [20]. For managing disk resources, there are several mechanisms by which disk-access bandwidth can be reserved [21, 22]. In parallel with this OS-level extension, we are developing a new QOS Manager by extending the QOS-Control Server.

Generally, a continuous-media session uses multiple resources that have some mutual relations.[5] We are therefore extending the QOS factor to represent the "QOS path," which indicates the possible path of QOS change among resources.

Our experiment omitted the mechanism by which each CM-Thread adjusts the QOS to meet its reserve. A user-level support library for such a mechanism is now being developed [23]. This library provides a new thread mechanism for continuous-media processing, named "Q-Thread." By using a Q-Thread, a user can specify the tolerable range of the period and computation time for invocations, as attributes of the thread. The Q-Thread library "translates" these attributes into the QOS factor, and registers this with the QOS-Control Server. The reserve issued by the server is "reverse translated" to some possible combination of period and computation time (on the basis of the user-specified QOS-control policy). In accordance with the period, a user-specified periodic entry point is called, and the calculated computation time is passed at this time as a hint for workload adaptation.

In the Keio-MMP project, we are proposing a Conductor/Performer architecture as a model for continuous-media processing on a microkernel [3]. In this model, a synchronization server (Conductor) sets up a media stream by "conducting" multiple media-specific servers (Performers), such as a storage server, an image server, a sound server, and a window server [24]. The QOS-Manager function will be integrated into the Conductor, and the Conductor will manage the allocation of resources to continuous-media sessions on the basis of the QOS-Ticket model.

The QOS Manager can be considered as a kind of scheduler based on the QOS factor. A user-level real-time thread package is now being developed as a part of our project [25]. It will be possible to use the scheduling mechanism from this package in a QOS Manager based on our QOS-Ticket model.

7 Related Work

There have been several studies of QOS-control architecture. Researchers at Lancaster University have tried to provide an integrated and coherent framework for QOS control [26, 27]. An orchestration mechanism for coordinating multiple connections with QOS control has been proposed by the present authors [28]. A group at the University of Pennsylvania has proposed a "QOS Broker" model that organizes multimedia resource management over network [29]. This model uses database files named "QOS profile" for the QOS translation, while our model uses the feedback and adaptation mechanism.

The Rialto OS by Microsoft Research proposes hierarchical resource management in the operating system [30]. The highest-level "Resource Planner" controls the resource allocation of the whole system. This mechanism resembles our QOS Manager, but the QOS Manager is implemented as a user-level server outside of the operating system. People in SRI International have incorporated a "benefit function" mechanism into their QOS Manager to decide the QOS control for multiple sessions [31]. The RT-Mach group at Carnegie Mellon University is also developing a server for QOS control and a network-phone system, using the processor capacity reservation.

[5] For example, it may happen that when the available disk bandwidth is small, only a small amount of the CPU resource is required.

For scheduling a processor in continuous-media processing, the Rate-Based Execution model has been proposed by a group at the University of North Carolina [32]. This model uses (x, y, d) parameters to represent the execution "rate" of a process. There have also been several studies of processor scheduling for continuous-media processing [33, 34].

For dynamic QOS control of each session, a group at Columbia University has proposed a rate-shaping mechanism for coded video data such as MPEG [35]. In addition, the IBM European Networking Center has proposed a media-scaling architecture with its transport system, HeiTS [36]. There has also been research on video players that can control the QOS dynamically [37].

In relation to these studies, we think that the unique features of our proposed mechanism are as follows:

- We focus on QOS control of multiple sessions.
- Our model incorporates both reservation and adaptation.
- Actual implementation is being done on RT-Mach.

Whereas some of the above research projects focus on the QOS control of individual sessions over a network, ours focuses on dynamic QOS control of multiple sessions in a system. However, it may be possible to use the QOS-control policies of the above projects for adjusting the QOS of each session in our QOS-Ticket model. Some of the previous QOS-control research considers only the resource reservation or the resource adaptation, while our QOS-Ticket model combines reservation and adaptation. We have also started to implement the QOS Manager and its support library in accordance with the QOS-Ticket model.

8 Conclusion

In this paper, we proposed a QOS-Ticket model as a new resource management and QOS control architecture for continuous-media applications. In this QOS-Ticket model, resource reservation and adaptation are combined, and dynamic QOS control is achieved through cooperation of an operating system, a QOS Manager, and individual continuous-media sessions. The QOS-Ticket, which represents the resource reservation for each session, mediates these activities.

A prototype of the QOS-Ticket model, named QOS-Control Server, has been implemented on RT-Mach, using that system's processor capacity reservation mechanism. This paper has also described an experiment with the prototype that shows the effectiveness of the QOS-Ticket model.

Acknowledgements

The authors would like to thank members of the Keio-MMP project and the Keio-IBM partnership program for their help and advice.

References

[1] Keio-MMP Project: WWW Home Page, URL: <http://www.mmp.sfc.keio.ac.jp/>.

[2] K. Kawachiya et al.: "Extending Real-Time Mach for Continuous Media Applications," *Collected Abstracts, 4th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 55–58 (1993).

[3] N. Nishio et al.: "Conductor-Performer: A Middle Ware Architecture for Continuous Media Applications," *Proc. 1st Intl. Workshop on Real-Time Computing Systems and Applications*, pp. 122–131 (1994).

[4] K. Kawachiya et al.: "Evaluation of QOS-Control Servers on Real-Time Mach," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 123–126 (1995).

[5] J. F. Koegel Buford et al.: *Multimedia systems*, Chapter 8, ACM Press SIGGRAPH Series, Addison-Wesley (1994).

[6] H. Tokuda et al.: "Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network," *Proc. ACM SIGCOMM '92*, pp. 88–98 (1992).

[7] T. Nakajima and H. Tokuda: "Implementation of Scheduling Policies in Real-Time Mach," *Proc. 2nd Intl. Workshop on Object-Oriented Operating Systems* (1992).

[8] K. Kawachiya et al.: "QOS Control of Continuous Media: Architecture and System Support," *IBM Research Report, RT0108*, IBM (1995).

[9] M. J. Accetta et al.: "Mach: A New Kernel Foundation for UNIX Development," *Proc. USENIX 1986 Summer Conf.*, pp. 93–112 (1986).

[10] D. Golub et al.: "Unix as an Application Program," *Proc. USENIX 1990 Summer Conf.*, pp. 87–95 (1990).

[11] H. Tokuda et al.: "Real-Time Mach: Towards a Predictable Real-Time System," *Proc. USENIX Mach Workshop*, pp. 73–82 (1990).

[12] H. Tokuda and T. Nakajima: "Evaluation of Real-Time Synchronization in Real-Time Mach," *Proc. USENIX Mach Symposium*, pp. 213–221 (1991).

[13] T. Kitayama et al.: "RT-IPC: An IPC Extension for Real-Time Mach," *Proc. USENIX 2nd Microkernel and Other Kernel Architecture Symposium*, pp. 91–104 (1993).

[14] H. Tokuda et al.: "A Real-Time Thread Model for Continuous Media Applications," *ART Group Tech. Report*, CMU, May 1993 (1993).

[15] H. Tokuda and T. Kitayama: "Dynamic QOS Control based on Real-Time Threads," *Proc. 4th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 113–122 (1993).

[16] C. W. Mercer et al.: "Processor Capacity Reserves: An Abstraction for Managing Processor Usage," *Proc. 4th Workshop on Workstation Operating Systems*, pp. 129–134 (1993).

[17] C. W. Mercer et al.: "Processor Capacity Reserves: Operating System Support for Multimedia Applications," *Proc. Intl. Conf. on Multimedia Computing and Systems*, pp. 90–99 (1994).

[18] C. Topolcic et al.: "Experimental Internet Stream Protocol, Version 2 (ST-II)," *Internet RFC-1190* (1990).

[19] L. Zhang et al.: "RSVP: A New Resource ReSerVation Protocol," *IEEE Network, Sep. '93*, pp. 8–18 (1993).

[20] S. Kihara et al.: "An Implementation of ST-II Protocol as a User-Level Server on Real-Time Mach," *Collected Abstracts, 4th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 59–62 (1993).

[21] R. L. Haskin: "The Shark Continuous-Media File Server," *Proc. IEEE COMPCON Spring '93*, pp. 12–15 (1993).

[22] H. Tezuka and T. Nakajima: "Design and Implementation of a Continuous Media Storage System on Real-Time Mach," *JAIST Research Report, IS-RR-94-15S*, JAIST (1994).

[23] K. Kawachiya and H. Tokuda: "Dynamic QOS Control of Continuous-Media Processing based on the "QOS-Ticket" Model," (in Japanese), *Proc. 7th IPSJ Computer System Symposium*, pp. 141–148 (1995)

[24] S. Tada: "Real-Time Extension of X Window System for Continuous Media Applications," *Proc. 1st Intl. Workshop on Real-Time Computing Systems and Applications*, pp. 294–298 (1994).

[25] S. Oikawa and H. Tokuda: "User-Level Real-Time Threads," *Proc. 11th IEEE Workshop on Real-Time Operating Systems and Software*, pp. 7–11 (1994).

[26] A. Campbell et al.: "A Multimedia Enhanced Transport Service in a Quality of Service Architecture," *Proc. 4th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 123–136 (1993).

[27] A. Campbell et al.: "Dynamic QoS Management for Scalable Video Flows," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 107–118 (1995).

[28] A. Campbell et al.: "A Continuous Media Transport and Orchestration Service," *Proc. ACM SIGCOMM '92*, pp. 99–110 (1992).

[29] K. Nahrstedt and J. M. Smith: "The QOS Broker," *IEEE Multimedia*, Vol. 2, No. 1, pp. 53–67 (1995).

[30] M. B. Jones et al.: "Support for User-Centric Modular Real-Time Resource Management in the Rialto Operating System," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 55–65 (1995).

[31] L. C. Schreier, and M. B. Davis: "System-Level Resource Management for Network-Based Multimedia Applications," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 127–130 (1995).

[32] K. Jeffay and D. Bennett: "A Rate-Based Execution Abstraction for Multimedia Computing," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 67–78 (1995).

[33] J. Nieh and M. S. Lam: "Integrated Processor Scheduling for Multimedia," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 215–218 (1995).

[34] R. Yavatkar and K. Lakshman: "A CPU Scheduling Algorithm for Continuous Media Applications," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 223–226 (1995).

[35] A. Eleftheriadis and D. Anastassiou: "Meeting Arbitrary QoS Constraints Using Dynamic Rate Shaping of Coded Digital Video," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 95–106 (1995).

[36] L. Delgrossi et al.: "Media Scaling for Audiovisual Communication with the Heidelberg Transport System," *Proc. ACM Multimedia '93*, pp. 99–104 (1993).

[37] K. Fall et al.: "Workstation Video Playback Performance with Competitive Process Load," *Proc. 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 179–182 (1995).