A Negotiation-Based Resource Management Framework for Dynamic QOS Control

Kiyokuni KAWACHIYA
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato,
Kanagawa 242, Japan
<kawatiya@trl.ibm.co.jp>

Hideyuki TOKUDA
Faculty of Environmental Information,
Keio University
5322 Endo, Fujisawa,
Kanagawa 252, Japan
<hxt@sfc.keio.ac.jp>

Abstract

To change the quality of service (QOS) of a continuous-media processing dynamically according to the system environment, a new resource management model is needed that supports resource negotiation between the application and the system by combining resource reservation and adaptation. Realization of such a model will require the solution of several problems, such as how to specify requests for mutually related multiple resources and for discrete QOS-controllable points, and how to allocate and calibrate resources to satisfy the specification. To solve these problems, this paper proposes a new resource management model that allows resource allocation and QOS adjustment over multiple continuous-media applications. In this model, each application can request or calibrate resources through an integrated abstraction named a “Resource-Ticket.”

Keywords: Dynamic QOS Control, Management of Multiple Resources, QOS Architecture, QOS Negotiation, Micro-Kernel, QOS-based API for Applications

1 Introduction

Changing the quality of service (QOS) of a continuous-media processing dynamically according to the system environment necessitates the use of not only a resource reservation mechanism, but also some adaptation mechanism based on feedback of actual resource consumption. Therefore, the following usual resource management techniques are insufficient:

1. No reservation or feedback
   A program cannot determine how much resource is available. The behavior of a continuous-media application is not predictable.
2. Resource reservation only
   A program can use resources preferentially within the reserved limit. Admission
   control and static QOS management are possible.

A new technique is required:

3. Resource negotiation, which combines reservation and adaptation
   The application adapts its processing to meet the allocated (reserved) resources by
   changing the QOS dynamically.

For this purpose, a new resource management interface (an abstraction for writing a
resource-aware application) must be provided.

As such a resource management model combining reservation and adaptation, we
have proposed a “QOS-Ticket” concept and evaluated its prototype for CPU resource
management [1, 2, 3]. However, while we were generalizing and extending the prototype,
several problems arose, of which the most important are how to support mutually related
multiple resources and how to support discrete QOS-controllable points. This paper first
describes these problems in detail, and then proposes ways of improving the resource
management model in order to solve the problems.

Section 2 describes our old QOS-Ticket model and the problems we have identified.
Section 3 introduces an improved “Resource-Ticket” abstraction to cope with the prob-
lems, and describes the functions of each component. Section 4 describes a prototype
implementation of the new resource management model on Real-Time Mach. After dis-
cussions on related and future work in Section 5, Section 6 offers some conclusions.

2 Background and Problems

The most notable characteristic of continuous-media data is the existence of timing con-
straints. To handle such data stably, it is necessary for an application to be able to
reserve system resources (CPU, memory, disk, network, etc.) for the processing. How-
ever, a reservation mechanism is not sufficient for dynamic QOS control, because only the
application knows how the QOS can actually be changed.

To control the QOS dynamically, some framework for developing a resource-aware ap-
plication is necessary. One solution is to specify a range for an application’s requirements
of each resource. The system allocates resources in the specified range, and the appli-
cation dynamically adapts its processing to match the allocated resources by changing
the QOS. As such a resource management framework, we have proposed a “QOS-Ticket”
model [1].

2.1 The QOS-Ticket Model

Figure 1 illustrates the structure of the QOS-Ticket model. This is a model for controlling
resource allocation and QOS adjustment among multiple continuous-media sessions. The
QOS management is achieved through the cooperation of an operating system, a QOS
Manager, and individual sessions. The “QOS-Factor” and “QOS-Ticket” mediate these activities.

In the QOS-Ticket model, as a rule, all the system resources are used with reservation. Each continuous media session registers its QOS-Factor with the QOS Manager when it is started. A QOS-Factor contains a priority and a tolerable allocation range (minimum and maximum values) for each resource request. The QOS Manager calculates the resource allocation 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 QOS-Ticket is a “ticket” for the session to use resources preferentially. It records information on actual resource consumptions through the ticket as well as the reservation status of the resources. By using this information as a hint, each session adjusts its own QOS to meet the resource restriction [2]. The guaranteeing, enforcement, and monitoring of resource consumption are performed by the operating system.

2.2 Several Problems in the Model

We have developed a prototype of the QOS-Ticket model, and performed several experiments for a CPU resource [3]. These experiments showed that the model is very effective for handling a single resource (CPU) and a continuously changeable QOS.

However, several problems became clear as we extended the prototype. Among them, the following major problems forced us to change the model:

1. Mutually related multiple resources

Usually, multiple resources have some mutual relations as regards their consumption. For example, to achieve high QOS services, not only more CPU resource
but also more of other resources is needed. As another example, to reduce the network bandwidth, more CPU resource may be needed for compressing the data. If resources are allocated individually without considering these correlations, they cannot be fully utilized by a session.

2. Discrete QOS-controllable points

In our old prototype, we assumed that the CPU consumption can be adjusted continuously by changing the period or processing amount of the session correspondingly. However, the QOS cannot always be changed continuously. When the screen resolution is changed in video processing, for example, the changeable points becomes discrete — full-size, half-size, quarter-size, etc. In such cases, odd resources that do not match the controllable points have no meaning and may be wasted.

Both these problems arise because we are focusing on dynamic QOS control. In a static QOS-control model where a QOS cannot be changed during a processing, the resource allocation occurs only at the time a session is invoked, and is not changed dynamically. Therefore, the resource management system does not need to know the relations among resources or QOS-controllable points.

The above-mentioned problems generate the following additional two problems:

3. Resource requesting and allocation technique

Our old prototype adopted tolerable ranges for specifying resource requests. But it is insufficient to specify the relations among resources and QOS-controllable points; a new request-specification method is also needed. In addition, a new technique is needed for allocating multiple resources to multiple sessions while satisfying the specified requirements.

4. Resource estimation and calibration

The resources needed for a specific processing operation may vary according to the hardware performance, media data, or other factors. It is difficult to estimate in advance what amounts of resources are needed for a processing operation. There must be some support for estimating the initial resource requests and calibrating them during the processing. This process must be carried out for all multiple discrete QOS-controllable points.

The next section describes how the resource management model was improved to solve the four problems.

3 Solving the Problems with a New Resource Management Abstraction

We have improved the resource management model to solve the four problems, using the following basic strategies:
• Change the resource-request method (for Problems 1 and 2)
• Develop a new resource allocation algorithm (for Problem 3)
• Improve a resource estimation/calibration algorithm (for Problem 4)

The following subsections describe these improvements in detail.

3.1 The Resource-Request Table

One solution worth considering for Problem 1 (multiple related resources) is to represent the relations by using some mathematical expressions. But in Problem 2 (discrete QOS-controllable points), it is not very significant that the resource allocation can be changed continuously. Therefore, we modified the model to specify multiple pairs of resource requests for multiple QOS-controllable points (named “QOS-Types”), rather than specifying tolerable ranges.

Table 1 shows an example of the new specification. In this table, three QOS-Types and their resource requirements are specified. If we switch from QOS-Type 1 to Type 3, all the resource requirements are reduced. This corresponds to reducing the frame rate in video processing. If we switch from Type 1 to Type 2, on the other hand, the CPU and memory requirements are increased while the network requirement is reduced. This corresponds to compressing data by using more CPU resource.

The “Q-Values” in the table indicate the satisfaction levels of the session when the corresponding QOS-Type is granted. This can also be considered as an indication of the actual quality, with a high value indicating high quality. The resource allocation algorithm discussed in Section 3.3 tries to grant a QOS-Type whose Q-Value is as high as possible.

Table 1: Resource-Request Table

<table>
<thead>
<tr>
<th>QOS-Type</th>
<th>CPU (%)</th>
<th>Memory (MB)</th>
<th>Disk (Mbps)</th>
<th>Network (Mbps)</th>
<th>Q-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>10</td>
<td>3</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>20</td>
<td>3</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>50</td>
</tr>
</tbody>
</table>

3.2 The Resource-Ticket Abstraction

By changing the resource-request specification from individual “ranges” to a “table,” we can deal with Problems 1 and 2. Figure 2 shows a new resource management model that incorporates this improvement. The basic structure is the same as that of the QOS-Ticket model shown in Figure 1, but the new model includes the Resource-Request Table, and is generalized to some extent.
In the new model, a “Resource-Ticket” is provided as an integrated abstraction through which each continuous-media session can request or calibrate resources.\footnote{This can be considered as a combined and generalized version of the QOS-Factor and QOS-Ticket in the old QOS-Ticket model.} This abstraction is the interface for writing a resource-aware application.

One Resource-Ticket is provided to each continuous-media session, and each session sets up its own “Resource-Request Table” in the ticket. On the basis of the tables of all the Resource-Tickets, the “Resource Allocator” decides on an allocation of resources that will satisfy as many sessions as possible (see Section 3.3 for details). The allocator reserves resources in accordance with the result, and writes into the Resource-Ticket the information on which QOS-Type is granted. This resource allocation is performed every time the tables are modified.

Each continuous-media session is executed with the granted QOS-Type. The operating system guarantees the use of resources within the specified limit. It also observes the actual resource consumption of each session, and these values are also written into the corresponding Resource-Ticket. On the basis of this feedback, each session calibrates the resource requirements and repeatedly modifies the Resource-Request Table (see Section 3.4 for details).

\footnote{This can be considered as a combined and generalized version of the QOS-Factor and QOS-Ticket in the old QOS-Ticket model.}
To implement the model, Problems 3 (resource allocation) and 4 (resource calibration) must be solved. The former is managed in the Resource Allocator, and the latter is managed by individual continuous-media sessions. The following subsections describe the behaviors of these components in detail.

### 3.3 Allocation of Multiple Resources to Multiple Sessions

On the basis of the Resource-Request Tables received from multiple sessions, the Resource Allocator decides on how to allocate resources in such a way as to satisfy as many sessions as possible. To be more precise, it tries to find an allocation pattern that maximizes the sum of the Q-Values of granted QOS-Types.

This problem is formulated as follows. Suppose that the Resource-Request Table of session \( i \) contains the following \( k \) patterns of resource requirements (QOS-Types) for resources \( A, B, \cdots \).

\[
\begin{array}{cccc}
\text{QOS-Type} & \text{Resource } A & \text{Resource } B & \cdots & \text{Q-Value} \\
1 & a_i(1) & b_i(1) & \cdots & q_i(1) \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
k & a_i(k) & b_i(k) & \cdots & q_i(k) \\
\end{array}
\]

When the total amounts of resources are \( A, B, \cdots \), the problem is expressed as an integer-programming problem of finding the values of \( x_i(j) \in \{0, 1\}, \sum_j x_i(j) \leq 1 \) (for all \( i \)) that satisfies \( \sum_{i,j} a_i(j)x_i(j) \leq A, \sum_{i,j} b_i(j)x_i(j) \leq B, \cdots \), and maximizes \( \sum_{i,j} q_i(j)x_i(j) \). Figure 3 shows the problem for the case in which there are two resources. After the problem has been solved, an index \( j \) that satisfies \( x_i(j) = 1 \) indicates the granted QOS-Type for session \( i \).

The above-mentioned problem subsumes the knapsack problem and is NP-complete [4]. If the resource request is quantized by using some unit (e.g. to \( M \) steps), the problem becomes NP-complete in a weak sense and can be solved in \( O(nkM^{n-1}) \) steps exactly by using a dynamic-programming approach, where \( n \) is a number of sessions and \( r \) is a number of resources. However, it still takes a lot of time for large \( M \) and \( r \), and usually it is better to introduce some heuristic algorithm, such as a local-search method based on Lagrange relaxation or randomized rounding [5].

The algorithm can be replaced without affecting other components such as Resource-Tickets, continuous-media sessions, and the operating system. It is also possible to introduce some resource allocation policy here.

### 3.4 Resource Estimation and Calibration

Using the feedback information in the Resource-Ticket, and possibly other information, each continuous-media session must estimate and calibrate its resource requirements. This calibration takes advantage of the iterative nature of continuous-media processing. In continuous-media processing, a procedure, such as retrieving and displaying a frame of
video data, is generally performed iteratively. It is therefore possible to specify some arbitrary resource estimations at the beginning and correct the Resource-Request Table according to the actual resource consumption after every iteration (or several number of iterations).\footnote{In MPEG video processing, the processing amount differs with the type of picture (I, P, or B). But in many cases, the calibration method can be used if the sequence from one I-picture to the next I-picture is regarded as one iteration.}

The basic structure of our resource negotiation algorithm in each session is as follows:

1. At first, the Resource-Request Table is empty. Start the negotiation with the QOS-Type that has the highest Q-Value (i.e., the most preferred QOS-Type).

2. Add the QOS-Type to the table. Specify zeros (or small values) for initial resource requests. As a result, the QOS-Type is granted by the Resource Allocator.

3. Perform an iteration with the granted QOS-Type. After the processing, correct (calibrate) the resource-request entry for the QOS-Type on the basis of the actual resource consumption information contained in the Resource-Ticket.
4. While the QOS-Type is still granted (i.e., while resources are allocated), continue the calibration by performing further iterations.

5. If no QOS-Type is granted, add to the table the QOS-Type that has the next-highest Q-Value, and continue the negotiation from Step 2.

In this algorithm, the negotiation starts with the QOS-Type that has the highest Q-Value. When resources for the requested QOS-Type cannot be granted, the next preferred QOS-Types are added incrementally. If the system environment changes during the processing and a higher QOS-Type is granted again, resource calibration for the QOS-Type restarts from Step 4.

Specifying zeros as initial resource requests in Step 2 means that resources are not guaranteed for the first iteration of each QOS-Type. If this becomes a problem, it is possible to specify some suitable values as the initial estimates by using data from the previous execution, for example, or from some database.

4 Prototype Implementation

Currently, a prototype of the Resource-Ticket abstraction is being implemented on the Real-Time Mach (RT-Mach) operating system [6], where several resource reservation mechanisms are already available [7, 8, 9]. Figure 4 shows the structure of the prototype. The functions of the Resource Allocator and Resource-Ticket abstraction are provided by a server [10] on RT-Mach named the “Resource Manager.”

Table 2 lists all the exported functions (APIs) of the Resource Manager. \texttt{rticket\_create()} and \texttt{rticket\_terminate()} are used to create and delete a Resource-Ticket in the manager. Resource requests to the ticket are manipulated through a structure named \texttt{resreq\_t} that represents one row of the Resource-Request Table. This structure includes the number of the QOS-Type, the requested amount of each resource, and the Q-Value. \texttt{rticket\_setreq()} adds, modifies, or deletes the request, while \texttt{rticket\_getreq()} retrieves the request from the ticket. \texttt{rticket\_setstat()} sets various options in the ticket. This function can be used for registering a port through which resource-allocation changes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{rticket_create()}</td>
<td>Create a Resource-Ticket</td>
</tr>
<tr>
<td>\texttt{rticket_terminate()}</td>
<td>Delete the Resource-Ticket</td>
</tr>
<tr>
<td>\texttt{rticket_setreq()}</td>
<td>Add/modify/delete a resource request</td>
</tr>
<tr>
<td>\texttt{rticket_getreq()}</td>
<td>Retrieve a resource request</td>
</tr>
<tr>
<td>\texttt{rticket_setstat()}</td>
<td>Set various states in the ticket</td>
</tr>
<tr>
<td>\texttt{rticket_getstat()}</td>
<td>Get various states from the ticket</td>
</tr>
</tbody>
</table>
are notified asynchronously, and for resetting the resource monitoring. `rticket_getstat()` is used to determine which QOS-Type is granted and what quantities of resources are actually consumed.

In the present implementation, the resource estimation and calibration are performed by each continuous-media session itself. However, it is desirable that some support library for negotiation and QOS translation should be provided, like in our old prototype [2].

5 Related and Future Work

The problems treated in this paper arise mainly as a result of the desire to dynamically control the resource allocation and QOS of multiple sessions that use multiple resources. Little work has been done on this specific requirement. Moser deals with a resource allocation problem for mutually related multiple resources, and introduces a heuristic algorithm using a local-search method based on Lagrange relaxation [11]. This algorithm can be applied in our Resource Allocator. However, there has been no discussion of
support for calibrating resource requests, which is the other focal point of our paper.

For estimating necessary resources, a “QOS Broker” model from the University of Pennsylvania proposes the use of database files named “QOS profiles” [12]. This approach can be incorporated into our model in order to estimate the initial values of resource requests (in Step 2 of the negotiation algorithm), although the main mechanism of our model is iterative calibration.

Several other QOS-control architectures have been proposed by various groups, including “QoS-A” from Lancaster University [13], “HeiTS” from IBM European Networking Center [14], and “NRP” from the University of Stuttgart [15]. These architectures mainly focus on end-to-end QOS control in distributed environments, rather than on multiple sessions or dynamic control.

In our model, resources are managed in a centralized manner by the Resource Allocator, and there may be some difficulty in extending the model to a distributed environment. One conservative solution is to continue to use centralized resource management in a distributed environment. As an aggressive solution, we are considering introducing a notion of “cost” into each resource, so that each session has to bid to acquire resources. In this solution, the Q-Values can be diverted for use as the bidding values.

Another possible extension of the Resource-Ticket is a support for applications other than continuous-media processing. For example, in a mobile-computing environment, the amounts of resources available (e.g., the communication bandwidth and processor power) change dynamically. The Resource-Ticket abstraction can be utilized for disclosing the resource status to mobile applications. The Odyssey project at Carnegie Mellon University has proposed such an interface for writing resource-aware mobile applications [16]. This interface uses ranges (“bounds of tolerance”) for requesting resources, like our old model. Resource allocation among multiple sessions or relations between multiple resources do not seem to be considered, because their target is an application in a mobile client.

6 Conclusion

To change the the resource allocation and QOS of multiple continuous-media sessions dynamically according to the system environment, a new resource management model that combines resource reservation and adaptation is necessary. To realize such a model, several problems must be solved, including how to handle multiple related resources and discrete QOS-controllable points. To solve these problems, we have proposed a new resource management model that uses an integrated abstraction named a “Resource-Ticket.”

Each continuous-media session registers its resource requirements for each QOS-Type in the Resource-Ticket as a Resource-Request Table. On the basis of the tables, the Resource Allocator decides how to allocate resources to sessions. Each session is carried out with the granted QOS-Type, and iteratively calibrates the table in accordance with the actual resource consumption. This paper has also investigated the resource allocation and calibration algorithms needed to implement the model.
Acknowledgments

The author would like to thank to Dr. Takeshi Tokuyama of IBM Tokyo Research Laboratory for his valuable comments on solving the knapsack problem of resource allocation.

References

Research Report

A Negotiation-Based Resource Management Framework for Dynamic QOS Control

Kiyokuni KAWACHIYA
IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242, Japan

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

Limited Distribution Notice
This report has been submitted for publication outside of IBM and will be probably copyrighted if accepted. It has been issued as a Research Report for early dissemination of its contents. In view of the expected transfer of copyright to an outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or copies of the article legally obtained (for example, by payment of royalties).