# An Interactive Interface and RT-Mach Support for Monitoring and Controlling Resource Management


Clifford W. Mercer and Ragunathan Rajkumar *

School of Computer Science

*Software Engineering Institute
Carnegie Mellon University
Pittsburgh PA 15213
cwm@cs.cmu.edu, rr@sei.cmu.edu



## Abstract

*Real-time applications in dynamic systems such as multimedia systems may change their timing characteristics on the fly and may be created and terminated at any time. A priori resource allocation decisions may not hold throughout the lifetime of such applications because of changing user needs and interests. The ability to monitor and control system resources dynamically is fundamental to building flexible and dynamic multimedia systems. In this paper, we describe the mechanisms and tools we have built for monitoring and controlling operating system resource reserves in Real-Time Mach. The system supports a resource reservation abstraction called processor capacity reserves, and an interactive tool named* rmon *uses these mechanisms to display the processor usage for each reserved activity in the system and to allow the user to change the processor reservation dynamically; it can also coordinate the reservation change requests with a separate quality of service manager which makes policy decisions regarding which requests are granted.*


## 1. Introduction

Real-time applications depend on the operating system to manage resources in a timely way. The timing constraints of these programs make them particularly sensitive to the performance characteristics of the hardware and software computing environment on which they are executing. For example, in multimedia systems, the bit rates achieved for an MPEG video decoder vary with processor speed [9]. Likewise, the bit rates achievable for a decoder running in isolation are greater than for a decoder running concurrently with some other computation. In other words, the functionality provided by these programs depends upon the limitations of the hardware platform and, in a multitasking environment, the capabilities of OS resource management algorithms.

Many systems, such as those designed for multimedia applications, are intended to support a mix of real-time programs which changes over time. Rather than having a fixed task set as in many embedded real-time systems, the task set and even the timing constraints associated with each task can vary as the user interacts with the system. The user may initiate or terminate programs, or change the quality of service (QOS) parameters of some programs as they run. QOS parameters, such as number of frames per second, resolution, and pixel depth for video, are constantly subject to change as the user's focus or activities change.

The OS must provide mechanisms to effect dynamic changes in QOS parameters. The decision to make any change must however be made by high-level layers based on information collected from the OS. Suppose that a user or other system policies can obtain precise information regarding the resource usage of concurrent multimedia programs. More resources can then be allocated to programs which need a better quality of service, or some resources can be reclaimed from programs which no longer need their current level of QOS. Mechanisms and tools for monitoring performance and controlling resource management can satisfy changing requirements as warranted by the programs' behavior and the changing focus of the user.

This paper describes the RT-Mach [14] framework to dynamically monitor and control resource usage and how it can be used. Section 2 provides some background on currently available tools for performance measurement and control, and Section 3 briefly describes the support for resource monitoring and control that is currently supported in RT-Mach. Section 4 presents an interactive tool named *rmon* which allows a user to interactively change resource allocations even on running programs. Section 5 explains the resource management environment for which the tool is intended. Section 6 discusses some related work. Finally, Section 7 presents some concluding remarks.

## 2. Background and Motivation

Some tools for monitoring and controlling system resources do exist but in rather primitive form. For example, Apple's MacOS on the Macintosh offers rudimentary "*Memory View*" and "*Memory Sizer*" commands which


This work was supported in part by U.S. Naval Research and Development under contract number N00014-94-1-0816, in part by the Office of Naval Research and in part by Loral Federal Systems. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of NRaD, ONR, Loral, or the U.S. Government.

display the current memory allocation to resident tasks and also provide a way to set memory limits for specific applications [1]. In addition, MacOS has a few other resource management features for sizing the disk cache and enabling virtual memory. However, it offers no information about or control over other resources such as processor time, network bandwidth, or disk bandwidth.

Most notably, resource adjustments (where available) are applicable only statically and not dynamically while programs are running. For example, the memory allocation to a program cannot be changed while it is running.

At another end of the performance monitoring spectrum, many sophisticated tools are available for design-time program and system performance analysis, primarily used to identify and rectify application performance bottlenecks. For example, tools like *gprof* [3] and the Performance and Coverage Analyzer (*PCA*) for VAX/VMS [2] provide support for some very detailed performance analyses. They enable a programmer to learn a lot about where application time is being spent and how various system resources are being divided among active processes. However, due to the fine granularity of the details involved, these otherwise powerful tools do not provide on-line performance analysis. While the information must of necessity be collected on-line, the core of the analysis for final presentation to the user occurs off-line. Finally, these tools are intended to be information providers but not controllers which directly affect resource allocation and allow the system performance to be changed in useful ways.

Ideally, a tool for monitoring and controlling resource management in a real-time operating system would combine a moderate level of detail with on-line performance monitoring as well as integrated control of resource management.

We shall describe in the next two sections the mechanisms currently implemented in RT-Mach and the user-level interactive tool, *rmon*. This tool, combined with the capabilities available in RT-Mach provide on-line monitoring and control.

## 3. OS Support for Resource Management

In Section 2, we mentioned that while some tools are capable of performing sophisticated system performance analysis, they are limited by the fact that operating systems typically provide very little resource usage information or offer very primitive support (if any) for controlling resource allocation. In this section, we discuss the resource reservation approach we are experimenting with in Real-Time Mach which provides the basis for the monitoring and control tools we have built. The presentation of this background material is brief, and a more detailed exposition can be found elsewhere [7].

We have designed a resource capacity reservation system for RT-Mach which (a) allows applications to request and reserve resource capacity, (b) enforces resource reservations, and (c) provides moderately detailed information about resource usage. The resource capacity reservation model extends to various types of resources such as processor time, memory, disk bandwidth, disk space, network bandwidth, and other I/O devices. We have implemented processor capacity reservation in RT-Mach.

The reservation model introduces a new *reserve* abstraction which represents reserved resource capacity. Each reserve has a type, such as a processor reserve or a network bandwidth reserve, and is associated with that resource in the system. Resource reserves can also be (dynamically) associated with threads. For example, a thread that is running under a particular processor reserve is assured that the CPU capacity represented by that reserve is available for the thread's consumption. Furthermore, this thread may pass its reserve along to any server that it invokes to enable that server to enjoy the benefits of running under the resource capacity reservation while performing its service for the thread. We refer to this as concept as "*reserve propagation*"[1]. The fact that the reserve is a first-class abstraction in the operating system enables this kind of capability.

The resource capacity reservation model consists of several components:

1. **a programming interface** allows applications to request reservations, terminate them, etc.
2. **an admission control policy** is used to decide when a reservation request can be accepted and when it must be denied.
3. **a scheduling algorithm** to schedule reserved applications such that their reserved capacity is available to them as appropriate.
4. **usage measurement and enforcement** to accurately measure the usage of reserved applications and to prevent them from interfering with each other or starving unreserved applications.

Brief descriptions of the above follow.

### 3.1. Programming Interface

The programming interface provides the means for applications to create reserves, destroy them, and request resource capacity to be associated with the reserve. Resource reservation requests are made in terms appropriate to the resource being reserved. For example, a processor capacity reservation is specified currently by a computation time and a period; the interpretation is that the computation time will be made available to this reserve for every period of real time. A processor reservation might therefore be for 30 ms

1. The definition of reserve propagation given here may sound analogous to priority inheritance [10]. While the two notions are indeed similar, they are not strictly analogous. In priority inheritance, a server or a blocking task inherits the priority of the highest priority task that it blocks. With reserve propagation, a server task would still inherit the priority of the highest priority task that it blocks. However, the amount of time spent by the server would be "charged" to the reserve of the client task for whom the service is actually being provided. Note that the charging is not done to the task whose priority is inherited (if different from the client task). This distinction forms the motivation behind our choice of the term "reserve propagation" instead of "reserve inheritance".

of computation time every 100 ms of real time. We refer to this period for reserve allocation as the reservation period.

### 3.2. Admission Control Policy

The admission control policy consists of an on-line scheduling analysis based on the reservation parameters that have been specified for a newly requested reserve and for all reserves already associated with that resource. The admission control policy is closely associated with the scheduling algorithm since the scheduler is responsible for guaranteeing the resource reservation. Critical zone analysis and/or a utilization bound is currently used for admitting processor reserves [4,5].

### 3.3. Scheduling Algorithm

The scheduling algorithm must ensure that the resource capacity that has been reserved is available to threads associated with that reserve. We currently use the rate monotonic scheduling algorithm [5] as a basis for our scheduler. The scheduler also plays a part in the enforcement of reservations and must determine when the thread or threads associated with a reserve have depleted the reserved capacity for a particular reservation period (not unlike aperiodic server algorithms [11]).

### 3.4. Usage Measurement and Enforcement

In order to inform the scheduler about whether the reserved capacity of a reserve has been depleted for a particular reservation period or not, the system must accurately measure resource usage on-line as it is being consumed. Once a reserve has been depleted for a reservation period, the system marks the reserve accordingly, and the scheduler can then recognize that threads associated with that reserve are no longer entitled to preferential treatment. The system must also mark the reserve as being “replenished” at the beginning of the next reservation when a new allocation of the resource capacity becomes available.

Since the operating system has these accurate measurements of resource usage, this information can easily be made available to the user level. To facilitate user-level tracking, RT-Mach not only keeps the accumulated resource usage, but also retains as a usage “checkpoint” the accumulated usage measured at the beginning of the current reservation period along with the timestamp of the measurement. As a monitoring aid, each reserve stores the last several of these checkpoints so that the recent history of the resource usage is always available for inspection from the user level. This also significantly reduces the overhead of monitoring since the number of calls to poll the system for this information is reduced.

## 4. rmon: A Reservation Management Tool

In this section, we describe a tool named *rmon* that we have designed and built on top of RT-Mach in order to provide the user with an interactive means to monitor and control the usage of system resources. The current version of *rmon* supports only processor capacity monitoring and control.

The following sections describe *rmon*’s monitoring features and control features in terms of processor capacity usage.

### 4.1. Usage Monitoring

*rmon* provides several ways of viewing information regarding processor reserves and their usage measurements. The primary view of *rmon* displays basic information about all processor reserves currently in the system. This information consists of a logical (application-defined) name associated with each reserve, a graphical representation of the recent usage information (normalized to the reservation period), and the reservation period itself. Figure 1 shows a screen dump of this primary view.

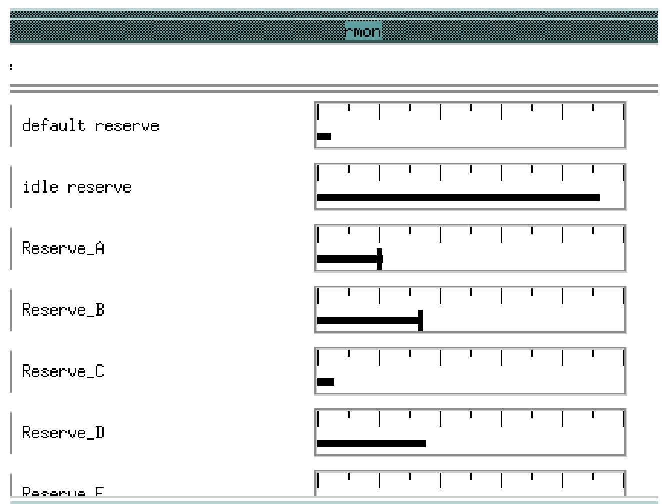


**Figure 1: rmon Main View**

The two system reserves called “*default reserve*” and “*idle reserve*” always exist, and thus always appear in the primary view. The default reserve is where all the usage for applications which do not have their own private reserves charge their usage. It has no actual resource capacity reservation associated with it and just accumulates the usage of the “unreserved” programs. The idle reserve accumulates the usage of the idle thread and also lacks an actual resource reservation.

For each reserve that has a resource capacity reservation associated with it, *rmon* displays a vertical bar in the usage graphic to indicate the level of the reservation, in terms of normalized capacity. In Figure 1, the reserves named *Reserve_A* and *Reserve_B* have resource capacity reservations associated with them whereas *Reserve_C* and *Reserve_D* do not. The vertical bars in the usage graphics of *Reserve_A* and *Reserve_B* indicate that they have reservations of 20% and 33%, respectively.

More detail is available for each reserve including a graphic of the recent history of resource usage as well as the parameters of the reserve. Figure 2 shows two examples of detail windows, one with a reservation and one without.


**Figure 2: rmon Detail Views**

As shown in the figure, the recent history graphic occupies the top portion of the window. It shows the normalized usage of the reserve over the last several reservation periods, and it advances in real-time in a manner similar to that of xload [8]. For the reserved activity, *Reserve_A*, this usage is fairly constant over time. For the unreserved activity, *Reserve_C*, this usage varies from period to period. Each window also displays the reserved computation time and the reservation period. The unreserved activity has a value of .100 seconds for the reserved computation time and for the reservation period. This indicates that usage measurements for this activity will be taken based on the reservation period, but that there is no actual resource capacity reservation.

### 4.2. Allocation Control

The level of the resource capacity reservation for a reserve, as indicated by the vertical bar in the normalized usage graphic (see Figure 1), can be dynamically changed by clicking the mouse in that usage graphic at the level desired for the reservation. This action modifies the reserved computation time parameter of the reserve without changing the reservation period.

As can be seen from the above description of *rmon*, it provides a moderate level of detail (more than xload for example whose view corresponds to that of each of *rmon*'s detail views), performs on-line monitoring and integrates a degree of resource allocation and control.

## 5. Quality of Service Management

The resource capacity allocation control aspect of the *rmon* tool is intended to form just one part of a more extensive resource management facility. This facility consists of a quality of service (QOS) manager to coordinate information about static user preferences, dynamic user preferences, and dynamic system load information (as illustrated in Figure 3).

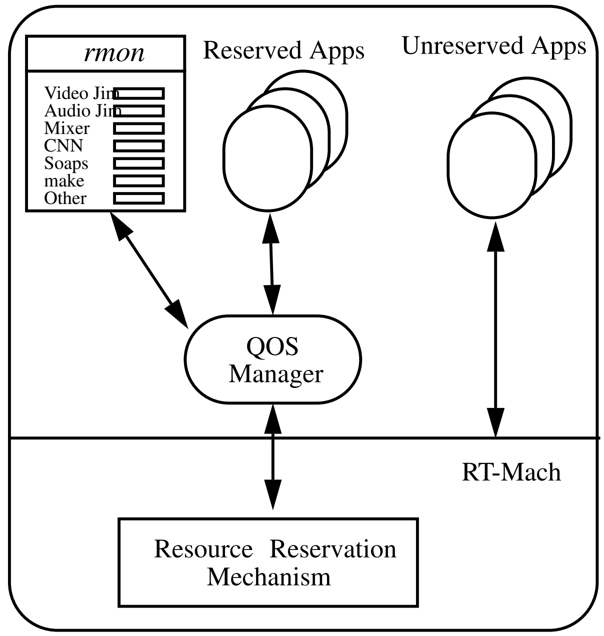


**Figure 3: Resource Management Schematic**

The QOS manager uses all of the information it gathers to make policy decisions about how to allocate resource capacity to various activities, and it uses the resource reservation abstraction to specify the resource allocation requirements at a relatively high level (compared to a lower-level mechanism like using a fixed priority assignment to try to control scheduling behavior).

The static user preferences used by a QOS manager might come from a configuration file located in the user's home directory or in a system default directory. Such a file could contain arbitrarily sophisticated rules for the QOS manager to use in making allocation policy decisions. For example, the file might contain rules to indicate how the user's focus should affect resource allocation, rules to determine which applications are more important (e.g. specifying that audio/video applications are more important than file transfer), rules about how temporal properties indicate which applications are more important (e.g. giving recently created applications preference over older applications), and rules about how past usage should affect future reservation.

The dynamic user preferences come from a tool like *rmon* or from a similar mechanism associated with a window manager. In either case, cues given by the user which can be picked up in the user interface are very important to the policy decisions that must be made about where to allocate resource capacity in the system. These cues can be explicit as in the case of the *rmon* tool where the user makes certain gestures to change the resource capacity allocations of various activities. Or the cues can be implicit as in the case of a window manager which notices which window has the user's focus (based on the position of the mouse pointer) and passes this information along to the QOS manager for use in making resource capacity allocation decisions.

Information about the recent resource usage of various activities might be used to determine what the future resource reservation levels should be for those activities. For example, if an audio player receiving transmissions over the network becomes quiet due to long-lasting lull at the sender, it may be appropriate to notice the lack of resource usage in the associated reserve and temporarily scale down the reservation level in order to free up more reservable capacity for other activities. A QOS manager with this feature should undoubtedly also provide a mechanism for such dormant applications to come back to life at their original reservation level once their activity begins to pick up again.

Finally, we note that when a QOS manager has the responsibility for allocating resource capacity for activities which use many different types of resources, a balance problem arises among those different resources. The problem of how to optimize resource reservation levels for multiple interacting resources is a difficult one which depends on exactly how the resources interact and how regular or predictable the activity itself is. At any rate, the QOS manager with its many sources of resource allocation preferences, its access to the resource capacity allocation mechanism based on the idea of reservation, and its access to fairly detailed resource usage information is poised to attack this problem. We implemented a couple of simple QOS managers on RT-Mach to gain some experience with using the reservation system in this context. These are briefly described below.

## 5.1. A Reactive QOS Manager

A QOS server, called ReactiveQOS, exports an interface that allows the programmer to create and terminate reserves, to request a reservation at a specific desired level, and to set preferences for the minimum reservation level. ReactiveQOS keeps track of the desired and minimum level, and if the reserved load becomes high and the server has difficulty granting desired reservation levels or even the minimum reservation levels for new requests, the server begins to downgrade some of the previously granted reservations to their minimum levels in order to admit the new reservation request at its minimum level. Thus, ReactiveQOS reacts to new reservation requests that strain the available resource capacity by trying to free up resource capacity from among previously reserved activities, subject to the limitations that those activities allow as expressed by their minimum reservation levels.

## 5.2. A Proactive QOS Manager

Another QOS manager, called ProactiveQOS, takes an extensive preference vector from each application which includes a maximum reserved level, a desired reserved level, a minimum reserved level, an adjustment priority and an adjustment granularity factor. It then attempts to provide to each QOS client at least its desired reserved level. The adjustment priority indicates the priority of the QOS client with respect to downgrading or upgrading its reserved level as the system load changes. If a new QOS request from an application to the QOS manager is denied by kernel admission control, the QOS manager will downgrade the reserve of the QOS client with the lowest adjustment priority first before trying to "steal" any resources from QOS clients with higher adjustment priorities. Similarly, if additional resource capacity becomes available (when applications terminate for example), QOS clients with higher adjustment priority will be upgraded first.

The adjustment granularity indicates to the QOS server how fine the changes in the application's thread ought to be. For example, a video application may want its reservation increased or decreased only in discrete steps perhaps with the steps equal to the processing time of one video frame. Any smaller increases or decreases may not be helpful to the client. A granularity of zero in contrast would indicate that even very minor adjustments to a client's reservation are permissible.

One of many adjustment policies can also be chosen within the ProactiveQOS manager. For example, all clients may be dropped to their desired reserved levels before the client with the lowest adjustment priority would be dropped to its minimum reserved level. Alternatively, the client with the lowest adjustment priority can be dropped to its minimum reserved level before any of the other clients are downgraded. By choosing a high adjustment priority, a zero adjustment factor along with identical values for the minimum, desired and maximum reserved levels, a QOS client can specify a constant "untouchable" requirement, i.e. a hard real-time constraint which once granted is guaranteed and enforced by the kernel[2].

Finally, this QOS server is proactive. It uses the reserve checkpoint information from the kernel to actually track how much of a QOS client's granted reservation is actually being used, and it can reclaim such unused resource capacity (up to the minimum reserved level) and grant it to other QOS clients which are currently bumping against their reserved levels (below their requested maximums).

2. It must be noted here that plain priority-driven scheduling approaches fail when a higher priority application enters an infinite loop. In contrast, the RT-Mach reservation system would preempt the application when its reservation runs out, and schedule other applications with valid reservations (until a finite replenishment to the preempted application's reserve occurs at a future point in time).

## 6. Related Work

Very few tools support on-line control of resource capacity allocation, much less dynamic on-line control as tasks using the resource are running. Tools like *xload* [8] provide a very simple view of the cumulative processor load on a workstation. It does this on-line, but it lacks any element of control. Other tools like the Memory Sizer on the Macintosh [1] offer control over a system resource, but the resource information is very simple, and one cannot set the memory size of a program while it is running.

Monitoring tools typically separate the functions of capturing performance data, analyzing the data, and having an effect on the system that was measured. The Advanced Real-Time Monitor (ARM) and Scheduler1-2-3 [12], originally designed for the ARTS Kernel [13] and more recently updated for RT-Mach, take this approach. ARM displays a scheduling history based on events captured by the kernel, but again the element of control of the monitored process is missing. The same is true of Scheduler1-2-3 which is a design tool that takes task set specifications as input and then analyzes the schedulability of those task sets. It is another example of an off-line analysis tool that can aid the design process but cannot provide on-line control over a running system.

Performance analysis tools such as *gprof* [3] and *PCA* [2] use PC sampling techniques to characterize program runtime behavior and object code to determine the static structure. These tools do not exercise control over program behavior directly.

## 7. Conclusion

On-line control of resource allocation based on resource usage monitoring allows flexibility and dynamism to be achieved when (a) applications are sensitive to availability of resources such as processing power and time available, (b) the task mix in a system is dynamic, and (c) the system is user-centric or policy-centric. The resource capacity reservation mechanisms of Real-Time Mach along with the *rmon* tool provide this crucial support for on-line monitoring of resource usage, dynamic control over the allocation of resources even while application programs run, along with a moderate level of information. A tool such as *rmon* can either make resource allocation policy decisions itself, via interactions from the user, or coordinate with a policy-based decision-maker, such as a quality of service manager. System overhead is certainly involved with these added support features, but our measurements indicate that these overheads are acceptable [6].

## Acknowledgements

The authors would like to express their appreciation to Jim Zelenka for his contributions to RT-Mach, and to Brian Hixon, Manish Modh and Bill Nagy for their work on the ProactiveQOS server.

## References

[1] Apple Computer, Inc. *Macintosh User's Guide*, 1991.

[2] Digital Equipment Corporation. *VAX Performance and Coverage Analyzer User's Reference Manual*.

[3] S. L. Graham, P. B. Kessler, and M. K. McKusick. gprof: a Call Graph Execution Profiler. In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, pages 120–126, June 1982.

[4] J. P. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[5] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. *JACM*, 20(1):46–61, 1973.

[6] C. W. Mercer and R. Rajkumar. Performance of an Interactive Interface for Monitoring and Controlling Resource Management in RT-Mach. Technical report, ART Project, Carnegie Mellon University, April 1995.

[7] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 90–99, May 1994.

[8] MIT. xload man page, 1988. X11 Window System, Release 4, Massachusetts Institute of Technology.

[9] K. Patel, B. C. Smith, and L. A. Rowe. Performance of a Software MPEG Video Decoder. In *Proceedings of the First ACM International Conference on Multimedia*, pages 75–82, August 1993.

[10] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[11] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *The Journal of Real-Time Systems*, 1(1):27–60, June 1989.

[12] H. Tokuda and M. Kotera. A Real-Time Tool Set for the ARTS Kernel. In *Proceedings of the 9th IEEE Real-Time Systems Symposium*, December 1988.

[13] H. Tokuda and C. W. Mercer. ARTS: A Distributed Real-Time Kernel. *ACM Operating Systems Review*, 23(3):29–53, July 1989.

[14] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Toward a Predictable Real-Time System. In *Proceedings of USENIX Mach Workshop*, pages 73–82, October 1990.