Metascheduling for Continuous Media

DAVID P. ANDERSON
Sonic Solutions

Next-generation distributed systems will support *continuous media* (digital audio and video) in the same framework as other data. Many applications that use continuous media need guaranteed end-to-end performance (bounds on throughput and delay). To reliably support these requirements, system components such as CPU schedulers, networks, and file systems must offer performance guarantees. A *metascheduler* coordinates these components, negotiating end-to-end guarantees on behalf of clients. The *CM-resource model*, described in this paper, provides a basis for such a metascheduler. It defines a workload parameterization, an abstract interface to resources, and an algorithm for reserving multiple resources. The model uses an economic approach to dividing end-to-end delay, and it allows system components to "work ahead," improving the performance of nonreal-time workload.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; C.4 [**Computer Systems Organization**]: Performance of Systems—*performance attributes*; D.4.1 [**Operating Systems**]: Process Management—*scheduling*; D.4.4 [**Operating Systems**]: Communications Management—*buffering*, *network communication*; D.4.7 [**Operating Systems**]: Organization and Design—*real-time and embedded systems*

General Terms: Algorithms, Design, Economics, Performance

Additional Key Words and Phrases: Multimedia, resource management

1. INTRODUCTION

We call audio and video *continuous media* (CM) because they are perceived as changing continuously over time. CM can be used in the user interface of computer systems. We say that a distributed computer system supports *integrated* CM if:

—The system stores and transmits CM data in digital form.
—CM data is handled by the same hardware (CPU, networks, I/O system) as other data.
—CM data is handled in the same software framework (operating system, file system, protocol stack) as other data.

This work, done at the University of California at Berkeley, was supported by NSF PYI grant CCR-86-57529, by IBM Corp., Sun Microsystems, and the California MICRO program.
Author's address: 1891 East Francisco Blvd., San Rafael, CA 94901.

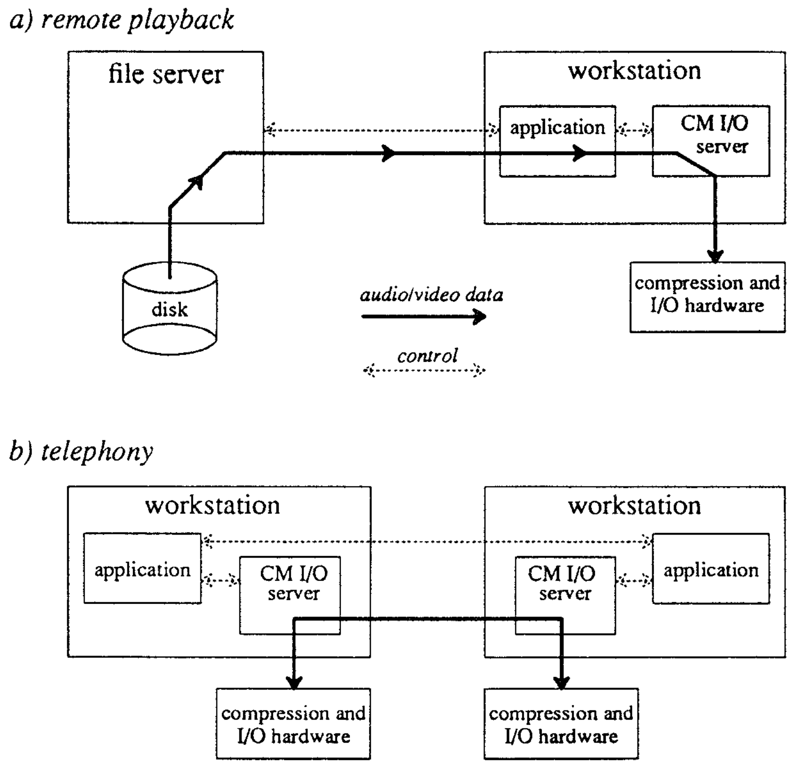

Fig. 1. *Remote playback* and *telephony* are typical applications in distributed systems that support continuous media.

Typical application programs in an integrated CM system, *remote playback* and *telephony*, are shown in Figure 1. Both programs convey CM data on network connections and use a *CM I/O server* [4, 7] to handle I/O at the user interface.

CM applications have performance requirements [16, 43]. One requirement is end-to-end throughput: in remote playback, for example, data must be transferred from disk to display hardware at a fixed average rate. This rate is determined by the data representation; for example, the rates of stereo CD-quality audio and DVI compressed video are 1.4 Mbps and 1.2 Mbps respectively [27]. A second type of requirement involves end-to-end delay: for telephony the upper bound on delay is typically in the range of 100–200 milliseconds, while musical applications (e.g., distributed rehearsal) may need delay as low as a few milliseconds [26].

Achieving CM performance requirements involves interrelated factors in the areas of hardware, OS mechanisms [15], and scheduling policies. This paper is concerned with the latter area. There are many approaches to scheduling for CM. For example, Hanko et al. [17] describe a "soft" model in which applications dynamically vary their data rates to adapt to system load. This model is well-suited to environments in which the demand on CM-related

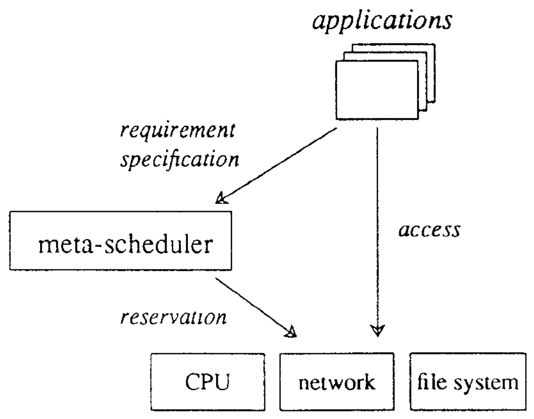

Fig. 2. A *metascheduler* acts as a mediator between applications and real-time system components. It reserves capacity on behalf of clients; the clients then access the components directly.

hardware regularly exceeds its capacity. For hardware-rich environments, and for applications like professional audio where a fixed quality must be maintained, a "hard" approach, such as that proposed by Jeffay [18], is more appropriate. In this approach, a CM application asks the system for a guarantee that its requirements will be met. The request may be rejected, in which case the application can either lower its requirements or wait until resources become available.

In order to make hard end-to-end guarantees, system components (CPU, file system, network, and so on) must provide a reservation mechanism. Compatible reservations must then be made through the entire chain of components. We call the agent that makes these reservations a *metascheduler* (see Figure 2). A metascheduler "reserves" components on behalf of client applications; it is not involved in the actual usage of the components.

A metascheduler must define a uniform model of how the components operate and interact. Design goals for this model include:

End-to-end semantics: CM application requirements apply to the entire data path from input device to output device. The model must therefore encompass all system components: CPU, storage, networks, and so on.

Flexible abstraction: The model should define an abstract interface for system components that accommodates a range of scheduling policies, allowing existing hardware and software components to be used with minimal modification.

Coexistence: The model should allow real-time and nonreal-time workload to coexist, and the impact of CM on the response time of nonreal-time traffic should be minimized.

The *CM-resource model* provides a basis for metascheduling. In the CM-resource model, clients reserve resources based on their worst-case needs, and resources offer hard performance bounds. However, unlike traditional hard real-time systems, the model allows resources to "work ahead", making more resource capacity available to bursty nonreal-time clients. The CM-resource model is an outgrowth of the DASH distributed system's *Real-time message stream* abstraction [1]. The model uses ideas of Rene Cruz [10, 11] and was developed by Martin Andrews, Robert Wahbe, Shin-Yuan Tzou, Ramesh Govindan and the author [3].

The paper is organized as follows. Section 2 defines the idea of *resources*, subsystems that store, manipulate, or communicate CM data and that may be reserved in *sessions* with workload, delay, and cost parameters. Section 3 defines and explores *compound sessions*. Section 4 gives an algorithm for establishing compound sessions. Section 5 describes techniques for scheduling a CPU and a token-ring network in conformance with the CM-resource model. Section 6 discusses related work, and Section 7 is the conclusion.

2. RESOURCES AND SESSIONS

The CM-resource model decomposes a distributed system into a set of *resources*. A resource may be a single schedulable device such as a CPU, or a complex subsystem such as a network. Resources provide a standard interface, described in this section, for reservation.

2.1 Describing Workload

The CM-resource model defines a parameterization of workload, i.e., the arrival process at a particular interface in the system. Workload is described in terms of discrete *messages* (units of work, typically blocks of CM data).

Definition. $N_I(t_0, t_1)$ denotes the number of messages arriving at interface I in the time interval $[t_0, t_1)$.

In the CM-resource model, arrival processes are described as follows:

Definition. A *linear bounded arrival process* (LBAP) is a message arrival process with three parameters:

M = maximum message size (bytes),
R = maximum message rate (messages/second), and
W = workahead limit (messages)

that, for all $t_0 < t_1$, satisfies

$$N_I(t_0, t_1) \le R|t_1 - t_0| + W. \tag{1}$$

This workload description was originally devised by Rene Cruz to analyze multiprocessor interconnection networks [10, 11].

The long-term data rate of an LBAP is MR bytes per second. The parameter W allows programs and devices to generate "bursts" of messages that would otherwise exceed this rate. This reflects messages that have arrived "ahead of schedule," *not* burstiness in the underlying data stream. The extent to which messages arrive ahead of schedule is quantified as follows:

Definition. The *workahead* $w(t)$ of an LBAP at time t is

$$w(t) = \max_{t_0<t}\{0, N(t_0, t) - R|t - t_0|\}. \tag{2}$$

We observe without proof that $w(t)$ is well-defined. Intuitively, $w(t)$ is the largest amount by which the rate R is exceeded during any time interval ending at t. More concretely, $w(t)$ increases by 1 on each message arrival, decreases with slope $-R$ at other times, and remains nonnegative (see Figure 3).

CLAIM 1. *$w(t) \leq W$ for all t.*

PROOF. The claim follows from Eqs. 1 and 2. □

A bound on arrivals during a time interval can be given in terms of workahead at its endpoints:

CLAIM 2. *For all $t_1 < t_2$,*

$$N(t_1, t_2) \leq w(t_2) - w(t_1) + R|t_2 - t_1|.$$

PROOF. From Eq. 2, let t_0 be such that

$$w(t_1) = N(t_1, t_0) - R|t_1 - t_0|.$$

Then

$$\begin{aligned} w(t_2) &\geq N(t_0, t_2) - R|t_2 - t_0| \\ &= N(t_0, t_1) + N(t_1, t_2) - R|t_1 - t_0| - R|t_2 - t_1| \\ &= w(t_1) + N(t_1, t_2) - R|t_2 - t_1| \end{aligned}$$

from which the claim follows. □

2.2 Describing Delay

It seems fair that if a message m arrives ahead of schedule at a resource and is queued there, the delay should be charged to the previous resource until the scheduled arrival time of m. We use a notion of delay that takes this into account. For a given LBAP, let $m_0 \cdots m_n$ denote the sequence of messages, and let $a_0 \cdots a_n$ denote their arrival times.

Definition. The *logical arrival time* $l(m_i)$ of a message m_i is

$$l(m_i) = a_i + w(a_i)/R.$$

Equivalently, $l(m)$ can be computed as follows:

$$\begin{aligned} l(m_0) &= a_0, \\ l(m_{i+1}) &= \max(a_{i+1}, l(m_i) + 1/R). \end{aligned} \tag{3}$$

Intuitively, $l(m)$ is the time m would arrive if workahead were not allowed.

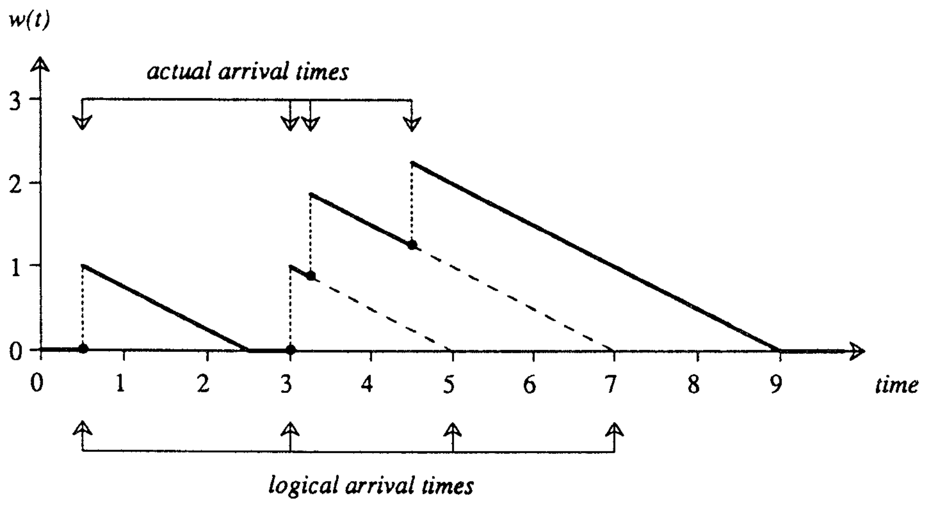

Fig. 3. The function $w(t)$ quantifies the extent to which arrivals at an interface are "ahead of schedule". It is used to compute *logical arrival times*. In this example, $R = 0.5$; note that logical arrival times are separated by at least $1/R$.

Definition. The *logical delay* $d(m)$ of a message m between two interfaces I_1 and I_2 is

$$d(m) = l_2(m) - l_1(m)$$

where $l_i(m)$ is the logical arrival time of the message at interface i.

The actual delay of a message m between two interfaces may be greater than $l(m)$ (if m arrives ahead of schedule) or less than $l(m)$ (if m is completed ahead of schedule). It can be shown that $l(m)$ is nonnegative.

2.3 Resources and Sessions

A *resource* handles streams of CM messages (and perhaps other workload as well). The messages arrive at an *input interface* and, when their processing by the resource is complete, at an *output interface*. A resource may be a single device (such as a CPU) or a set of interacting devices (such as a network).

A client must make a reservation with a resource prior to using it. These reservations, called *sessions*, have the following parameters:

M	= maximum message size (bytes),
R	= maximum message rate (messages/second),
W_{in}	= input workahead limit (messages),
W_{out}	= output workahead limit (messages),
D	= maximum logical delay (seconds),
A	= minimum actual delay (seconds), and
U	= minimum unbuffered actual delay (seconds).

The arrival process at the input interface must obey the LBAP parameters M, R and W_{in}, and the arrival process at the output interface must obey the LBAP parameters M, R and W_{out}. D is an upper bound on the logical delay between the input and output interfaces. A is a lower bound on the actual delay. U is a lower bound on the actual delay during which the message is not stored in host memory (this is nonzero for network resources, in which it represents propagation delay).

Each resource exports a procedural interface of the following form for creating sessions:

```
reserve(
  input: maximum message size and rate
  output:
    success/failure flag
    session ID
    maximum logical delay
    cost function
    minimum actual delay
    minimum unbuffered actual delay
)
relax(
  input: session ID, new maximum logical delay
)
free(
  input: session ID
)
```

These functions are used as follows:

(1) A call to `reserve( )` requests a session with the given workload parameters and the smallest possible logical delay bound.
(2) If it can accept the session, the resource returns the minimum possible logical delay bound D_{min} for the session, and reserves enough of its capacity to provide this bound. It also returns a *cost function* $C(D)$ (see below) whose value at D is the cost per unit time of a session with delay bound D.
(3) The caller chooses a specific delay bound $\bar{D} \geq D_{min}$, and calls `relax( )` to increase the delay bound to $\bar{D}$.
(4) The resource can then be used, subject to the session parameters. When this usage is finished, `free( )` deletes the session.

The details of the interface vary between resource types. For example, a CPU resource's `reserve( )` operation has a "maximum CPU time per message" parameter instead of the maximum message size, and a network resource is given the peer address. The interface applies to resources that handle CM data, either by processing it or by transporting it. Other types of resources serve as sources or sinks of CM data. These include file systems (disk drives and their associated software) and transducers (analog/digital conversion units, video display devices, etc.). We model such resources as providing a single input or output interface, with associated LBAP parameters. Delay is not involved, so the reservation interface exported by such resources is simpler than the one shown above.

The cost functions given by `reserve( )` are used to divide end-to-end delay between resources (see Section 4.1). Cost functions are nonincreasing: for a given throughput, a session with a smaller delay bound is more "costly" because it limits the resource's ability to accommodate other low-delay sessions. The functions may reflect real cost (e.g., for public networks) or load-balancing preferences. For example, the cost function for the CPU of a personal workstation might reflect the value to the owner of having "slack" in the CPU scheduling.

Note that the interface does not refer to workahead limits. We assume that resources on a host share a common pool of buffer memory, so workahead limits are relevant only at the interface between hosts (i.e., in network resources); see Section 3.3.

3. COMPOUND SESSIONS

In the remote playback application of Section 1, data is read from a disk, traverses a CPU, a network, and another CPU, and is consumed by decompression and display hardware. In the CM-resource model, such a situation is represented as a "compound session" (see Figure 4):

Definition. A *compound session* S is a sequence of sessions $S_1 \cdots S_n$ in which the output interface of S_i is the input interface of S_{i+1}.

The resources in a compound session handle a stream of messages in pipeline fashion. All the sessions must have the same throughput limit, and the output workahead limit of session S_i cannot exceed the input workahead limit of session S_{i+1}. The input interface of S is that of S_1, and the output interface of S is that of S_n. The logical delay d of a message m in S is the difference in logical arrival time of m at these two interfaces. Logical delays, and hence logical delay bounds, are additive in compound sessions:

CLAIM 3. *Let d and S be as above. Then $d \leq \sum_{i=1}^{n} D_i$, where D_i is the logical delay bound of S_i.*

PROOF. Let l_i be the logical completion time of m in S_i, and l_0 be the logical arrival time in S_1. Then

$$\begin{aligned} d &= (l_n - l_0) \\ &= (l_n - l_{n-1}) + (l_{n-1} - l_{n-2}) + \cdots + (l_1 - l_0), \\ &\leq D_n + \cdots + D_1. \quad \square \end{aligned}$$

3.1 Eliminating Starvation Due to Jitter

Suppose a client transfers messages from the output interface I_{out} of a compound session to an output device that requires data at periodic intervals. The device is said to "starve" if, at some point, data is not available when it is needed. Jitter (variation in delay) can cause starvation even if long-term throughput is sufficient. We now show that the receiver can eliminate this starvation by delaying the start of output.

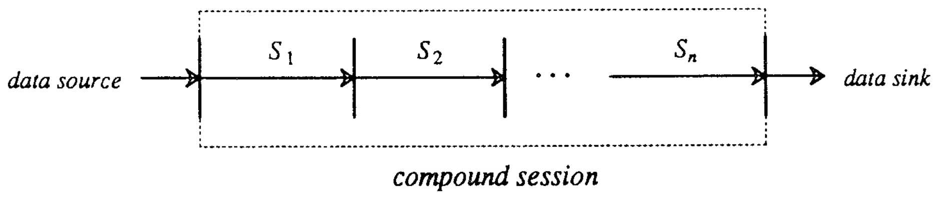

Fig. 4. A *compound session* is a sequence of sessions $S_1 \cdots S_n$ in which the output interface of S_i is the input interface of S_{i+1}.

Definition. The client is said to be *conservative* if it buffers data and waits until time $t_1 = t_0 + D$ to start output, where t_0 is the arrival time of the first message at the input interface I_{in} of S, and D is the delay bound of S.

Definition. An arrival process is *workahead-positive* if $w(t) > 0$ for all $t \in (a_0, a_n]$ (recall that a_i is the actual arrival time of message m_i).

CLAIM 4. *If an arrival process is workahead-positive then for any* $t \in (a_0, a_n)$

$$N(a_0, t) > R|t - a_0|. \tag{4}$$

PROOF. Suppose otherwise. Then there is some $t > a_0$ such that

$$N(a_0, t) \le R|t - a_0|.$$

Let t_1 be the least such t. Let $t_2 \in (a_0, t_1)$. Then $N(t_2, t_1) < R|t_2 - t_1|$ since otherwise we would have $N(a_0, t_2) \le R|t_2 - a_0|$, contradicting the minimality of t_1. Now from Eq. 2, we have $w(t_1) = 0$, contradicting the assumption that the arrival process is workahead-positive. □

CLAIM 5. *If the arrival process of a compound session is workahead-positive at its input interface, and the receiver is conservative, then the device never starves.*

PROOF. Suppose otherwise. Then there is a time $t_2 > t_1$ at which starvation occurs, i.e.,

$$N(t_0, t_2) < R|t_2 - t_1|. \tag{5}$$

Let $t_3 = t_2 - D$ (see Figure 5). Then $w(t_3) > 0$, so by Eq. 4, $N_{in}(t_0, t_3) > R|t_3 - t_0|$. All messages arriving between t_0 and t_3 are completed between t_0 and t_2, so $N_{out}(t_0, t_2) \ge N_{in}(t_0, t_3)$. Combining these inequalities with $t_3 - t_0 = t_2 - t_1$, we get $N_{out}(t_0, t_2) \ge R|t_2 - t_1|$, contradicting Eq. 5. □

A slightly more general result holds: Suppose the source of data has a variable rate not exceeding R. If the sender stays "ahead of schedule" relative to this variable rate, and the receiver waits until t_1 to start displaying, starvation will not occur.

To do conservative output, the receiver must wait until at least $t_0 + D$ to begin display. If the compound session spans multiple hosts, it may be difficult for the receiver to know exactly what time (of its local clock)

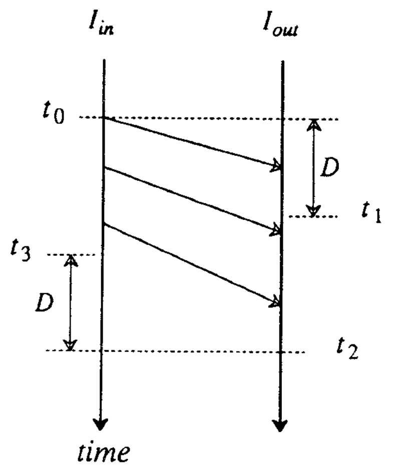

Fig. 5. Diagram for the proof of Claim 5. The receiver delays output until t_1, and is assumed to starve at t_2. This contradicts the assumption that the input interface is workahead-positive.

corresponds to t_0. There are several possible approaches that guarantee conservatism:

(1) Assume that all clocks are synchronized within ε. The sender timestamps the first message (i.e., includes t_0 in the message). The receiver then delays output until $t_0 + D + \varepsilon$.

(2) Assume that the delays of the network links on the path from sender to receiver are known. The first message has a "total delay" field D_{total}, initially zero. Just prior to sending the first message, each host adds its local delay, plus the delay of the outgoing link, to D_{total}. The receiving host delays output by $D - D_{total}$ after it receives the first message.

(3) In the absence of the above assumptions, the receiver can delay output by $D - A_{\min}$, where $A_{\min}$ is the sum of the minimum actual delays of the resources in S.

The extra buffer space required for conservative output (beyond that needed to accommodate incoming workahead and delay, to be discussed in Section 3.2) is εR in case 1, zero in case 2, and $R(D - A_{\min})$ in case 3.

3.2 Buffer Space Requirements

We now compute a bound on the buffer space in a host H used by a compound session S. Let $X_1 \cdots X_n$ be the resources in S for which messages are buffered in the main memory of H (see Figure 6). The incoming network (if any) is not in this list, since its output interface corresponds to message arrival in host memory. Let U be the minimum unbuffered time of X_n, let D_i be the logical delay bound of X_i, let $D = \Sigma_i D_i$, let W be the incoming workahead limit of X_1, and let R be the maximum message rate of S.

Claim 6. *The maximum number of messages buffered in host H for session S is $W + R(D - U)$.*

Proof. At a given time t, let m be the session's oldest message in host memory. Let a be the arrival time of m at X_1, and let $w(a)$ be the workahead

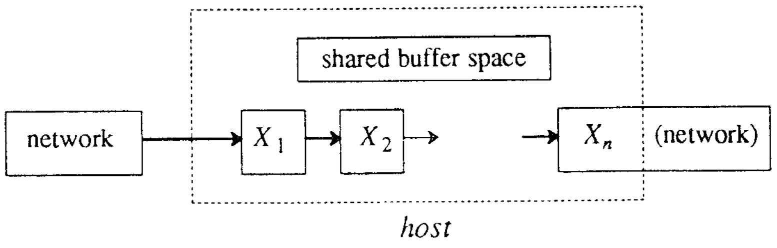

Fig. 6. A compound session includes resources $X_1 \cdots X_n$ in a given host. The resources use a common pool of buffer space for queued messages.

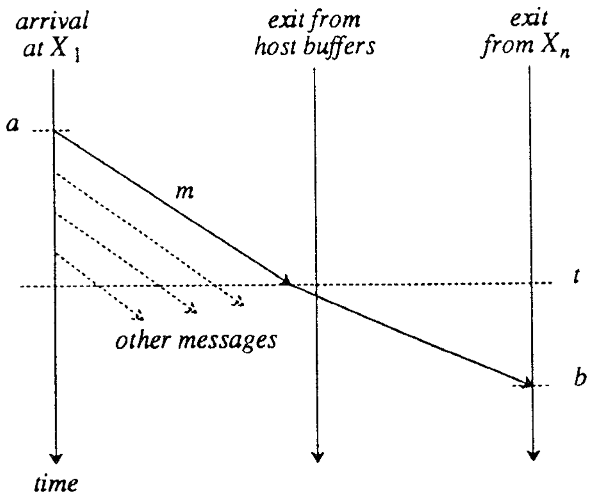

Fig. 7. Diagram for the proof of Claim 6. The number of messages buffered at time t is $N(a,t)$, where a is the arrival time of the oldest message in memory at time t.

at X_1's input interface at time a. The number of messages buffered in the host at time t is $N(a,t)$, i.e., the number of arrivals to X_1 in $[a,t)$ (see Figure 7). From Claim 2 we have

$$\begin{aligned} N(a,t) &\le w(t) - w(a) + R|t-a| \\ &\le W - w(a) + R|t-a| \end{aligned} \tag{6}$$

since $w(t) \le W$. Let b be the actual completion time of m in X_n, $w(b)$ the workahead at the output interface of X_n at time b, and $l(m)$ the logical delay of m in $X_1 \cdots X_n$. Then

$$\begin{aligned} D &\ge l(m) \\ &= (b + w(b)/R) - (a + w(a)/R) \\ &\ge b - (a + w(a)/R) \end{aligned}$$

so

$$b - a \le D + w(a)/R. \tag{7}$$

Now from the definition of U,

$$t \leq b - U. \tag{8}$$

Combining Eqs. 7 and 8 gives

$$t - a \leq D + w(a)/R - U. \tag{9}$$

Combining equations Eqs. 6 and 9 gives

$$\begin{aligned} N(a,t) &\leq W - w(a) + R(D + w(a)/R - U), \\ &\leq W + R(D - U). \quad \square \end{aligned} \tag{10}$$

This bound is realized in the case where arrivals consist of an initial group of W simultaneous messages followed by one message every $1/R$ seconds, and each resource uses its full delay for each message.

3.3 Limiting Workahead by Regulation

If a resource incurs variable delay, its outgoing workahead may exceed its incoming workahead. However, it is possible for a resource to reduce its outgoing workahead by doing *regulation*, i.e., by delaying outgoing messages that are completed ahead of schedule. Regulation may be necessary in compound sessions, since otherwise the workahead limit (and hence buffer space requirements) would increase without bound in the downstream direction. If resources on a host share buffer space, the total host buffer space requirement depends only on the workahead bound into the first resource in the host. Limiting the workahead between resources within a host does not affect buffer space requirements, so it need not be considered. Regulation is therefore necessary only in network resources.

The `reserve( )` operation of a network resource returns an outgoing workahead limit, namely the smallest possible workahead limit that the resource can provide. The `relax( )` operation takes a new outgoing workahead limit, which cannot be less than the existing limit.

4. ESTABLISHING COMPOUND SESSIONS

In creating compound sessions, the following issues must be addressed:

—How should a given bound on end-to-end delay be divided among resources? (It may not be fair, or even possible, to divide it uniformly.)

—It is desirable to set workahead limits as high as possible, given buffer space limits. How can this be done?

These issues are dealt with in the remainder of this section.

4.1 Compound Cost Functions

The cost of a compound session is the sum of the costs of its component sessions. The division of delay among the component sessions should minimize this cost. This *minimal-cost delay problem* can be formulated as follows.

Given a set of cost functions $C_1 \cdots C_n$ and an end-to-end delay bound D, find per-resource delay bounds $D_1 \cdots D_n$ that solve

$$\min \sum_{i=1}^{n} C_i(D_i)$$

subject to

$$\sum_{i=1}^{n} D_i < \overline{D}.$$

CLAIM 7. *The minimal-cost delay problem is NP-hard.*

PROOF. We show this by reducing the NP-complete PARTITION problem [19] to the minimal-cost delay problem. An instance of the PARTITION problem is as follows: Given a finite set A and for each $a \in A$ a size $s(a) \in Z+$, is there a subset $A' \subseteq A$ such that $\Sigma_{a \in A'} s(a) = \Sigma_{a \in A - A'} s(a)$? Consider the following instance of the minimal-cost delay problem: for each $a \in A$, let C_a be the function

$$\begin{aligned} C_a(d) &= s(a), \qquad d < s(a), \\ &= 0, \qquad d \geq s(a) \end{aligned}$$

and let $\overline{D} = \frac{1}{2}(\Sigma_{a \in A} s(a))$. Let $d(a)$, $a \in A$ be a solution to this instance of the minimal-cost delay problem. Assume without loss of generality that, for all $a \in A$, either $d(a) = s(a)$ or $d(a) = 0$. Let $C = \Sigma_{a \in A} C_a(d(a))$. Then $C = D$ iff there is a solution to the original PARTITION problem. □

The C_i produced in Claim 7 are piecewise constant. To make the minimal-cost delay problem tractable, we require that cost functions have the following property:

Definition. A cost function C is *tractable* if it is (1) piecewise linear with a finite number of vertices; (2) monotonic decreasing, and (3) convex (that is,

$$C(\alpha d_1 + (1 - \alpha) d_2) \leq \alpha C(d_1) + (1 - \alpha) C(d_2)$$

for all $\alpha \in [0, 1]$ and all $d_1 < d_2$).

A tractable cost function is defined on an interval $[d_1, d_2]$, where d_1 is the smallest delay bound that the resource can provide. Beyond d_2, the "cost" of extra buffer space exceeds the cost saved by the looser delay bound.

If cost functions are tractable, then the optimal delay assignment for a given total delay can be obtained in an amount of time proportional to the number of segments in the C_i.

Definition. Suppose tractable cost functions $C_1 \cdots C_n$ corresponding to resources $R_1 \cdots R_n$ are given. The *compound cost function* C, a piecewise linear function in which each segment is labeled with a resource name, is defined by the following procedure (see Figure 8).

(1) Sort the segments of $C_1 \cdots C_n$ in order of increasing (less negativ.) slope.

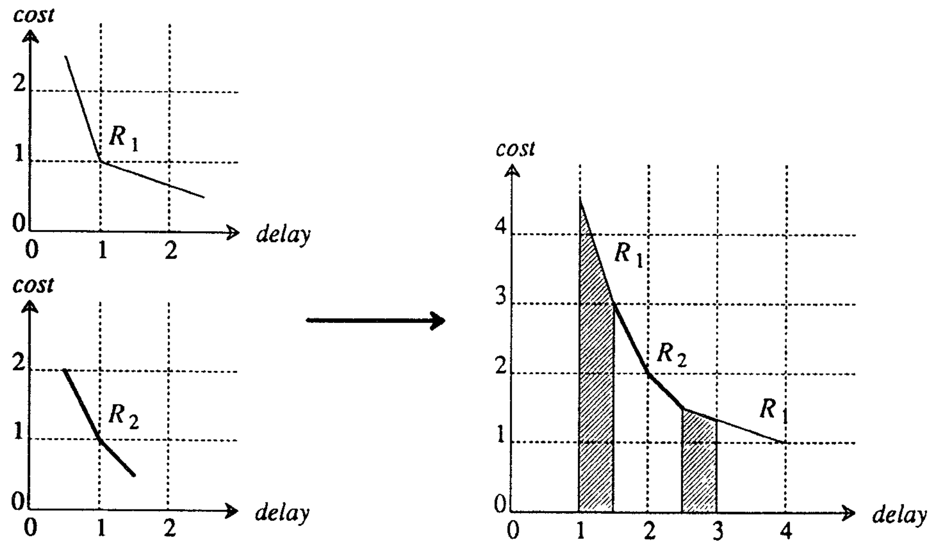

Fig. 8. The cost functions for several resources can be combined to form a *compound cost function* whose segments are labeled with the names of the resources. The function $F(C, R, d)$ returns the optimal amount by which to relax the reservation of resource R given a total delay bound d. In this example, $F(C, R_1, 3) = 1$, as shown by the shaded areas.

(2) Position the first (steepest) segment so that it begins at the sum of the upper left endpoints of the C_i.

(3) Position the remaining segments so that each begins where the previous ends.

Definition. Given a compound cost function C, a resource R, and a delay d, let $F(C, R, d)$ be the length of the set

$$\{x \le d : \langle x, C(x) \rangle \text{ is in a segment labeled with } R\}$$

(see Figure 8).

CLAIM 8. *Let d_i denote the least delay for which C_i is defined. Then the delay assignment given by $D_i = d_i + F(C, R_i, \overline{D})$ solves the minimal-cost delay assignment problem.*

The proof is simple; we omit it for brevity.

4.2 Compound Session Establishment

The CM-resource model defines an algorithm for establishing compound sessions. The algorithm is a protocol between sending and receiving clients, resources, *host resource managers* (HRMs), and memory managers on each host (see Figure 9).

The protocol has two phases. In phase one, a *request message* traverses the hosts from the source towards the sink. The request message contains the following items.

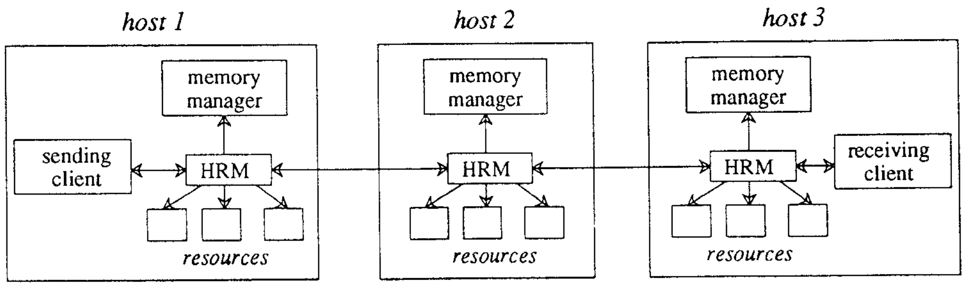

Fig. 9. The compound session establishment protocol involves clients, host resource managers (HRMs), memory managers, and resources.

—The sequence of hosts and resources involved, and an identifier for the receiving client.

—The message size S and rate R.

—End-to-end logical delay requirements: a *target* and *maximum* value, denoted E_{target} and $E_{\max}$. The goal of the algorithm is to establish a compound session with a logical delay bound as close to E_{target} as possible, and no greater than $E_{\max}$; and, given this delay bound, to minimize the cost of the session.

—The compound cost function C, the sum D of the delay bounds, and the sum A of the minimum actual delays, of the resources traversed so far.

—A workahead limit W.

A client initiates the protocol by passing a request message to its local HRM. In this message, C is empty and D and A are zero; the other fields are set by the client. On receiving a request message, a HRM does the following ($R_1 \cdots R_n$ denote the local resources involved in the compound session):

(1) Call `reserve( )` on $R_1 \cdots R_n$ in any order or in parallel.
(2) Reserve buffer space according to Eq. 10.
(3) If any reservation request fails, or if the total delay exceeds $E_{\max}$, free all reservations and return a failure message.
(4) Prepare an outgoing request message with D, A, and C updated according to the results of the `reserve( )` operations. The outgoing workahead W is returned by the `reserve( )` operation on R_n. Send this message to the HRM in the next host or, if this is the last host, to the receiving client.

The receiving client, on receiving the request message, selects an end-to-end delay E_{actual} for the session. The *excess logical delay* E_{excess} is given by

$$E_{excess} = \max(0, E_{target} - E_{actual}).$$

Phase two proceeds in the reverse direction. A *reply message* containing E_{excess} and a workahead limit W is passed back towards the source. The receiving client initiates phase two by passing a reply message to its local

HRM. On receiving a reply message, a HRM does the following:

(1) Reserve additional buffer space according to a host-specific policy; say the total (including the phase-one reservation) is B bytes.
(2) For each resource R_i, let $x = F(C, R_i, E_{excess})$, where C is the cost function computed by this HRM In phase one. Reduce x if needed so that total buffer space needs (see Eq. 10) do not exceed B. Call `relax( )` on R_i, increasing its delay bound by x. Subtract x from E_{excess}.
(3) Compute the largest value of W such that buffer space needs do not exceed B. Prepare a new reply message, containing W and the new E_{excess}. If this is the first host, pass this message to the sending client. Otherwise, pass it to the previous HRM.

At a given host, the period between phase one (reservation) and phase two (relaxation) has nonzero duration. Suppose a request for a second session S_2,with overlapping resource requirements, arrives during this period. The request for S_2 could be needlessly rejected (that is, it would have been accepted had it arrived after phase two). To avoid this situation, the following policy can be used. If a reservation request fails while a compound session establishment is in progress, the request is put in a FIFO queue of requests for that resource, and retried when a `relax( )` is done. If the request fails when all relaxation is finished, it is rejected.

Suppose a compound session request reserves resources R_1 and R_2, and a second request reserves the same resources but in a different order. If there is enough capacity for only one of the two sessions, both sessions could be rejected. This possibility can be eliminated by ordering the resources on a host, and reserving them in that order.

The algorithm described here minimizes total cost if buffer space is available to accommodate the optimal delay bounds. A more complex algorithm can minimize total cost given limits on buffer space in each host. This algorithm would require propagating information about the sequence of hosts, their available buffer space, and the locations of resources.

4.2.1 *Multicast and Other Communication Topologies.* The CM-resource model does not directly provide multicast. However, compound sessions can be combined to reflect situations in which data streams split or merge. The session establishment algorithm described above provide two "hooks" for this purpose: (1) the sending client request can include information about resources that "precede" the new session: (2) the receiving client, after getting a request message from the HRM, can do whatever it wants (including establishing more sessions) before it replies.

As an example, consider an application in which a stream of CM data is sent to multiple receivers. This can be done efficiently using a *multicast tree* [12]. If each edge of this tree is a compound session, the result is a *tree-structured compound session* that guarantees the throughput and delay to each of the recipients (see Figure 10).

The algorithm for establishing a tree-structured compound session is as follows. A tree of *multicast agents* $X_1 \cdots X_n$ is given. The root is the data

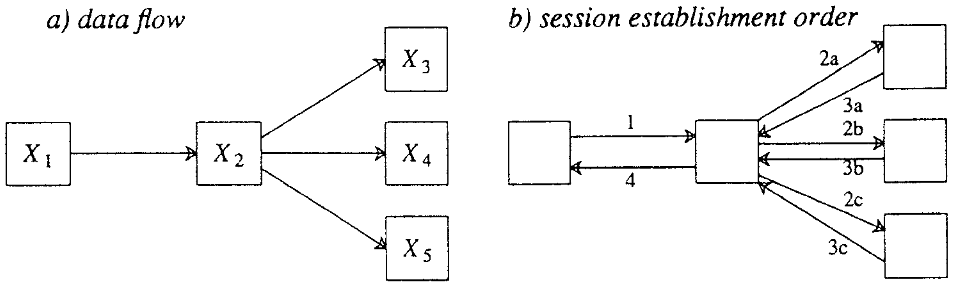

Fig. 10. A simple example of a tree-structured session. Multicast agent X_2 receives CM data from X_1 and forwards it to X_3, X_4 and X_5. When establishing the session, the order of messages is shown in b; (2a, 2b, 2c) and (3a, 3b, 3c) occur in parallel.

source and the leaves are data sinks. Values E_{target} and $E_{\max}$ are given for the end-to-end delay bound. The goal of the algorithm is to achieve a delay bound no greater than $E_{\max}$, and as close to E_{target} as possible, along each branch of the tree, and to minimize cost given these bounds.

Each agent receives a session request message M from the local HRM (the root agent X_1 receives its request from the client). For each child X_i, the agent calls the HRM, requesting a compound session traversing the CPU and network resources to X_i. The cumulative cost functions and delays for this request are taken from M. The agent then waits for replies for all children. Let D be the largest of the delays to the children, W the minimum of the workaheads to the children, and E the minimum of the excess delays. The agent forms a reply message containing D, W, and E, and passes this reply message to the local HRM.

A similar approach handles situations where data streams merge, e.g., a process receives data streams from several sources and uses a DSP processor to mix the streams. In both examples, dynamically extending the nonlinear session (e.g., to handle new participants in a conference or new data streams to be mixed) is possible, but may not give an optimal delay assignment.

4.3 Compound Sessions in the Internet

As an example of how the CM-resource model impinges on a standard network protocol hierarchy, we now describe the Session Reservation Protocol (SRP), a realization of the compound session establishment protocol for IP networks [2]. Using SRP, a compound session can be created and associated with a connection of TCP or any other upper-level protocol. The performance guarantees of the compound session apply to the data sent in one direction on the connection.

SRP acts as a host resource manager (HRM), but only for certain resources. On the sending host, SRP reserves the outgoing network resource. On a gateway, SRP all resources (CPU and outgoing network). On the receiving host, SRP makes no reservations, and simply conveys the session request to the receiving client. The clients of SRP are responsible for reserving all other resources that handle the CM data, such as input and output devices and

CPU on the end nodes. Thus, on the sending and receiving hosts the HRM function is divided between the clients and SRP.

SRP does not involve changes to existing protocols. However, it requires that a connection follow a static route through the network, so that the set of resources is fixed. This in turn requires modifying IP implementations to do static routing for packets that are part of a session. In addition, the kernel on a receiving host must associate packets with sessions (typically at the IP level) in order to correctly prioritize the handling of packets by the CPU.

5. IMPLEMENTING RESOURCES

We now discuss how typical resources (a CPU scheduler and a token-ring network) might provide the interface specified by the CM-resource model. A third example, a file system, is presented elsewhere [6].

5.1 CPU as Resource

Assume there is a single processor and that for each session S there is a process P that does all the work for S and no other work. The input interface for S is defined by message arrivals; if P is handling incoming messages from a network connection, arrivals occur when the network interface requests a receive interrupt. Message completion occurs when P makes a call indicating that it has handled the packet.

To exploit the advantages of a workahead in the CM-resource model, we propose the *deadline-workahead* scheduling policy. This policy classifies processes as follows. A *real-time process* is one that is associated with a session. At time t, a real-time process is called *critical* if it has an unprocessed message m with $l(m) \leq t$ (i.e., m's logical arrival time has passed). Real-time processes that have pending work but are not critical are called *workahead* processes. There are two classes of nonreal-time processes: *interactive* (for which fast response time is important) and *background*. The deadline-workahead policy is as follows.

—Critical processes have priority over all others, and are preemptively scheduled by earliest deadline (the deadline of a process is the logical arrival time of its first unprocessed message plus the delay bound of its session).

—Interactive processes have priority over workahead processes (but are preempted when those processes become critical).

—Workahead processes have priority over background processes. They need not be scheduled by earliest deadline; the policy might instead try to minimize context switches or address space switches.

Nonreal-time processes can be scheduled by time-slicing or any other general-purpose policy. Govindan and Anderson [15] describe *split-level scheduling*, an implementation of deadline-workahead scheduling that minimizes system call overhead.

The `reserve( )` operation exported by the CPU scheduler takes an extra input parameter: maximum CPU time per message. Suppose that a new session S has been requested, and that sessions $S_1 \cdots S_n$ already exist. The smallest possible delay bound D for S can be determined as follows. D is initially zero. The operation of the CPU is simulated under a "worst-case" workload [25]: all sessions generate maximum periodic workload starting at the same time. If the simulation reaches a point where no processes are runable, D is the desired delay bound. If a simulated message completion violates its delay bound, D is increased by an amount sufficient to eliminate the violation, and the simulation is restarted.

For this scheme to work, system software must be structured in a way that allows CPU execution to be matched with the correct session and message. For example, incoming protocol handling must be done by processes, rather than by a software interrupt routine as in current BSD UNIX-based systems [32]. When real-time processes run at the user level, they must correctly report deadline changes to the kernel. There are several possible sources of "priority inversion" in CPU scheduling: interrupt-handling overhead, interrupts-masked periods, priority inversion during locking (including kernel nonpreemption), etc. Methods for bounding these periods [40] can be used, and the reservation algorithm modified accordingly.

5.2 FDDI Network as Resource

FDDI is a 100 Mbps token-ring network [38]. The FDDI media access protocol has two data priorities. *Asynchronous* data uses a conventional token-passing protocol. *Synchronous* data has higher priority: If a given station has not seen the token within the Target Token Rotation Time (TTRT) then it may not send asynchronous data. Each station i has an allotment A_i for synchronous data; it may send at most A_i bytes each time it gets the token. The allotments can be changed using a Station Management (SMT) protocol. SMT ensures that

$$P + \left(\sum_{i=1}^{n} A_i \right) \Big/ B \le TTRT$$

where B is the data rate of the network and P is the propagation time around the ring. The maximum token rotation time is then 2(TTRT), the average token rotation time is TTRT [39], and each station is guaranteed a synchronous data throughput of

$$R_i = \frac{A_i}{TTRT} \tag{11}$$

bytes per second.

A session in an FDDI network "resource" is a point-to-point connection between two hosts. Message arrival at a session occurs at a call to a `network_send( )` routine in the sender. Completion occurs when a receive interrupt request is generated in the receiver. Two issues must be addressed: how to guarantee performance, and how to limit workahead. We discuss these separately.

5.2.1 *Scheduling and Reservation.* There are two levels of scheduling of the network medium: (1) contention by hosts for the medium and (2) scheduling of outgoing packets within a particular host. The policy for (1) is specified by the FDDI protocol; we propose using an earliest-deadline-first policy for (2).

The algorithm for `reserve( )` is as follows. The host uses SMT to request the largest possible synchronous allotment $\bar{A}$. If $\bar{A}$ is not sufficient for this host's sessions (both existing and requested) by Eq. 11, the new session is rejected. Otherwise, simulation is used to determine the least possible delay bound D. The system is simulated under worst-case conditions (all sessions begin simultaneously, 2TTRT elapses before the token arrives initially, and TTRT elapses between subsequent token arrivals). If a deadline is missed during the simulation, the delay bound D is increased by the smallest amount that would result in a different transmission order, and the simulation is restarted. The minimum unbuffered delay returned by `reserve( )` is the propagation time to the destination host. The workahead limit is RD, where D is the maximum queueing delay within this host (computed as above); this is the smallest workahead limit that can be guaranteed, given the regulation scheme described below.

The `relax( )` operation uses simulation to find the least possible allotment A given a new delay bound. A is initially zero. Whenever a deadline is missed, A is increased by the smallest amount that allows a packet to be transmitted in an earlier round, and the simulation is restarted. The resulting allotment is registered with SMT. Likewise, `free( )` computes the minimum allotment for the new set of sessions and registers it with SMT.

5.2.2 *Regulation of Packet Transmission.* Network resources may need to regulate packet transmission (see Section 3.3). This can be done as follows. When a packet is passed to `network_send( )`, the outgoing workahead of the session S is computed according to Eq. 3. If the result exceeds the workahead limit for S, the packet is enqueued in a separate *delay queue*, and a per-session *delayed-send timer* is set for a time when the outgoing workahead will fall below the limit. When the timer expires, packets on the delay queue are sent or queued for sending.

6. DISCUSSION AND COMPARISON

This section defends the design decisions in the CM-resource model and relates the model to existing work.

6.1 Design and Decision Rationale

The design of the CM-resource model was guided by the following assumptions:

(1) Resource capacity (CPU cycles, network bandwidth) usually exceeds application requirements.
(2) Applications have delay bound requirements that are usually either very low (teleconferencing) or moderately high (remote playback).

(3) Most hosts have enough extra buffer space to store a few seconds of CM data.
(4) Most CM data types (audio and video encodings) have a fixed, or at least a bounded, data rate.
(5) All system components can make real-time guarantees.

These assumptions are feasible for future distributed systems based on Gigabit networks and high-speed workstations with real-time CPU scheduling. They do not hold for systems based on components that are nondeterministic (Ethernet, UNIX) or low-capacity (ISDN). Other CM-oriented scheduling models, such as that proposed by Hanko et al. [17] are more appropriate in such cases.

Taking the above assumptions as given, we made design decisions in the following areas.

Reservationism. The CM-resource model is grounded in the idea of reservation: 100 clients can reserve 1 Mbps sessions in a 100 Mbps network, and the 101st is turned away. Other models would give each client 100/101 Mbps, and require them to adjust their data rates accordingly. There are advantages to each approach; arguments for "reservationism" include:

(1) Some applications (e.g., professional audio) absolutely require fixed performance levels.
(2) In future consumer applications (e.g., video-on-demand) it is likely that customers will be willing to pay for guaranteed quality levels.
(3) Dynamic data rate variation adds complexity. In applications that involve playback from storage it may be impractical to change data rates at the source.

One common argument against reservationism is that resource capacity is wasted when reservations are not fully used. This argument does not apply to the CM-resource model; unused resource capacity is unavailable for other reservations, but can be used by nonreal-time traffic or by workahead real-time traffic.

It would be possible to combine the two approaches by using reservation for a "base" performance level (for low-frequency information), and using a dynamic approach for high-frequency information that enhances but is not vital to the CM data stream.

Data rate model. Many CM data representations are constant-rate. These include uncompressed digital music (in which there are no silences) and video compression schemes such as DVI. However, some CM streams have rate variation; examples include silence-suppressed voice conversations [8], and variable-rate compressed video [35]. The CM-resource model requires that data streams have a bounded rate, and it makes reservations based on this rate. We chose not to model rate variation because resource overbooking (based on variable-rate data) implies packet loss and/or unbounded delays, contrary to our goal of low losses (see the following).

Workahead and logical delay. While CM data streams must have a bounded "underlying" rate, arrivals at an interface within the system may have an arbitrarily high rate over short periods. This burstiness is due to workahead and is reflected in the W parameter in the LBAP model. The LBAP model is minimally restrictive: it limits workahead to avoid buffer overrun, but it makes no other demands on the workload (e.g., there is no minimum intermessage gap).

Many real-time systems assume that delay bounds are "backlog-avoiding": the delay within a resource is bounded by the message interarrival time [25]. The CM-resource model is more flexible than backlog-avoiding models in two ways:

—The delay bound can differ from the interarrival time. In high-delay applications like remote playback it is important to allow delay bounds to exceed the interarrival time. In low-delay applications like telephony, the delay bound in a resource may have to be less than the interarrival time: Suppose a stream contains 33-millisecond frames and traverses 10 resources. In the backlog-avoiding model, its delay bound would be at least 330 milliseconds, which may be several times too large.

—The bounds apply to *logical* (not actual) delay. If the input has worked ahead, the actual delay may be greater than the logical delay bound. This flexibility can improve the response time of bursty nonreal-time traffic, as shown in Figure 11.

We used a hard upper bound on logical delay, rather than a statistical bound (i.e., 90% of messages are delayed less than X) to provide deterministic performance; for many CM applications, lateness is equivalent to loss.

Message loss and corruption. The CM-resource model is designed to eliminate data loss due to buffer overrun or lateness. Since the bit error rate of future fiber-optic networks will be very low [42], it may be reasonable to assume that errors in distributed compound sessions will occur at about the same rate as in processors and memory. At the very least, error-correcting codes, rather than retransmission protocols, can be used to provide required data integrity [30].

This *low-loss model* is a worthy goal. If significant error rates were allowed, low-delay applications (such as teleconferencing) would have to be able to deal with missing data, since retransmission is not viable in this case. Some data representations (e.g., differentially compressed video) do not tolerate data loss or corruption. High-delay applications that use these representations would have to use a retransmission protocol, at the cost of increased overhead and complexity.

Flow control, jitter, and synchronization. The CM-resource model requires that each network node "regulate" data transmission so that receiver buffers never overflow. This eliminates the need for flow control protocols in most cases (a possible exception is pausing during remote playback, but this could be handled at higher levels).

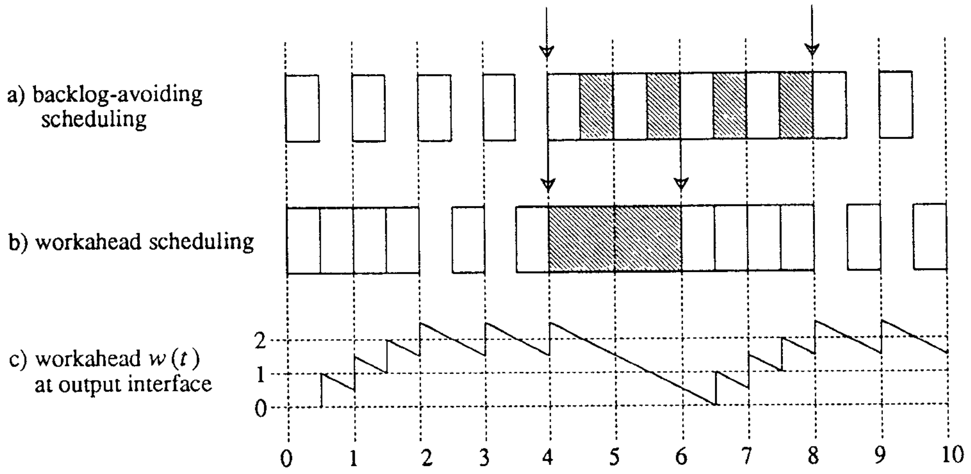
a) backlog-avoiding scheduling
b) workahead scheduling
c) workahead $w(t)$ at output interface
2
1
0
0
1
2
3
4
5
6
7
8
9
10

Fig. 11. A comparison of workahead and backlog-avoiding scheduling. A CPU is handling a stream of CM data (light boxes) that uses 50% of the CPU capacity A two-second nonreal-time task (dark boxes) arrives at time 4. Workahead scheduling is 2.5 seconds ahead of schedule at this point and can handle the new task in single uninterrupted CPU burst. With backlog-avoiding scheduling, the new task takes 4 seconds to complete.

Jitter (delay variation) is not inherently undesirable: it is the by-product of workahead scheduling. Isochronous (jitter-free) models force resources to be restrictively scheduled, as described above. The jitter on a compound session is bounded by its delay bound. This allows starvation-free output by buffering at the receiver, as described in Section 3.1.

The CM-resource model has no requirement for clock synchronization or a global time source, even if a resource (such as a network) spans multiple hosts. The workahead function can be computed based on local (unsynchronized) clocks. However, these local clocks must run at approximately the same rate. If the ratio of clock rates has an upper bound α, then reservations must be made for a throughput of αR, where R is the underlying data rate.

Many CM applications have synchronization requirements, e.g., that two streams must be displayed simultaneously or that one must immediately follow another [24, 34]. Some proposals have suggested providing synchronization functions in the network protocol hierarchy [41]. However, we believe such functions are better left to CM I/O servers [4]. The CM-resource model supports this by allowing the servers to know when to start I/O on a group of parallel synchronized streams (see Section 3.1).

The economic analogy. The CM-resource uses the idea of "cost" to describe how delay is divided among resources. This invocation of economics may seem spurious: the objective functions may be based on load, for example, not money. However, the economic analogy is important for several reasons:

(1) Future large-scale systems will include components, such as public data networks, that charge real money.

(2) Dissimilar resources may have no common basis for measuring load.
(3) By extending the economic model to include "benefit" functions (e.g., CD-quality audio with 50 millisecond delay is worth X dollars per second to the user) we can balance resource usage against the utility or importance of the workload.

6.2 Related Work

Much work has been devoted to the design of system components (networks, processing, storage) with real-time semantics. In the area of CPU scheduling, optimal uniprocessor policies are known [13, 25], and heuristic algorithms for more complex situations (multiple processors, resource contention, etc.) have been proposed [23, 45]. Issues of disk layout and scheduling are discussed in [33, 37]. Real-time LAN media access protocols are surveyed in [21]. The goal of end-to-end internetwork connections with meaningful "quality of service" parameters is discussed in [36]. The idea of logical arrival time was independently developed by Zhang [44], who applies it to fair queueing of data streams in network gateways.

Economic approaches have been used for resource allocation problems such as load-balancing [28], network access [22], and file placement [20]. More recently, research has been directed towards systems in which agents are selfish and competitive, and prices vary with demand [14, 29].

Finally, the idea of metascheduling is present in some distributed real-time systems [31, 45]. This work differs from ours because it focuses on the end-to-end latency of request/reply operations, rather than on the throughput and delay of data streams.

7. CONCLUSION

The CM-resource model is a basis for *metascheduling* in distributed systems that support integrated continuous media. The model includes a workload model (linear bounded arrival processes), an abstraction of reservable subsystems (resources), and a protocol for negotiating the parameters of end-to-end resource configurations. We have shown that the abstract interface can be implemented for a range of system components (CPU, network, file system). Because the model allows system components to work ahead, bursty nonreal-time traffic performs better. The model gives hard bounds on end-to-end delay, allowing starvation-free output. Because the amount of buffer space needed on a given host is known, buffer overrun can be eliminated, simplifying error management.

The CM-resource model might be extended in several ways. Mechanisms could be added to respond incrementally to changing conditions (resource costs, buffer space, or client requirements), or to allow sessions to be reserved "in advance" for specific periods. The economic model could be extended to handle factors other than the division of delay. In the current model, throughput and delay bounds are fixed by the client. Instead, "benefit functions" could be associated with CM data types and delay limits, and economic principles (i.e., maximizing benefit minus cost) could be used to determine data types, message sizes, and other parameters. More generally, economic principles could be used to determine the very structure of the application: for example, to choose between alternative topologies for distributed audio mixing [5, 9].

ACKNOWLEDGMENTS

The CM-resource model is a group effort. Martin Andrews and Robert Wahbe devised the basic model (LBAPs, resources, and compound sessions). Ralf Herrtwich, Shin-Yuan Tzou, Ramesh Govindan, Eric Barr, and Steve Lucco contributed substantially, as well.

REFERENCES

1. Anderson, D. P. A software architecture for network communication. In *Proceedings of the 8th International Conference on Distributed Computing Systems* (San Jose, Calif., June 1988).
2. Anderson, D. P., Herrtwich, R. G., and Schaefer, C. SRP: A resource reservation protocol for guaranteed-performance communication in the Internet. Tech. Rep 90-006, International Computer Science Institute, Feb. 1990.
3. Anderson, D. P., Tzou, S., Wahbe, R., Govindan, R., and Andrews, M. Support for continuous media in the DASH System. In *Proceedings of the 10th International Conference on Distributed Computing Systems* (Paris, May 1990), 54–61.
4. Anderson, D. P., and Homsy, G. A continuous media I/O server and its synchronization mechanism. *IEEE Computer 24*, 10 (Oct. 1991), 51–57.
5. Anderson, D. P., and Chan, P. Toolkit support for multiuser audio/video applications. *Comput. Commun.* (Oct. 1992).
6. Anderson, D. P., Osawa, Y., and Govindan, R. Real-time disk storage and retrieval of digital audio and video. *ACM Trans. Comput. Syst.*, to appear.
7. Angebrannndt, S., Hyde, R. L., Luong, D. H., Siravara, N., and Schmandt, C. Integrating audio and telephony in a distributed workstation environment. *Proceedings of the 1991 Summer USENIX Conference* (Dallas, Tex., Jan. 21–25, 1991), 419–434.
8. Brady, P. T. A statistical analysis of on-off patterns in sixteen conversations. *Bell Syst Tech. J. 47*, 1 (Jan. 1968).
9. Chow, C. Resource allocation algorithms for multimedia multiparty connections. Tech. Rep. EAS-CS-92-1, Univ. of Colorado at Colorado Springs, Jan. 1991.
10. Cruz, R. L. A calculus for network delay and a note on topologies of interconnection networks. Ph.D. dissertation, Rep. UILU-ENG-87-2246, Univ. of Illinois, July 1987.
11. Cruz, R. L. A calculus for network delay. *IEEE Trans. Inf. Theory 37*, 1 (Jan. 1991).
12. Deering, S. E., and Cheriton, D. R. Multicast routing in datagram internetworks and extended LANs. *Trans. Comput. Syst. 8*, 2 (May 1990), 85–110
13. Dertouzos, M. L. Control robotics: The procedural control of physical processes. *IFIP Congress* (1974), 807–813.
14. Ferguson, D., Yemini, Y., and Nikolaou, C. Microeconomic algorithms for load balancing in distributed computer systems. In *Proceedings of the 8th International Conference on Distributed Computing Systems* (San Jose, Calif., June 1988), 491–499.
15. Govindan, R., and Anderson, D. P. Scheduling and IPC mechanisms for continuous media In *Proceedings of the 13th ACM Symposium on Operating System Principles* (Pacific Grove, Calif., Oct. 14–16, 1991), 68–80.
16. Gruber, J. Performance considerations for integrated voice and data networks, *Comput. Commun. 4*, 3 (June 1981), 106–126.

17. HANKO, J. G., KUERNER, E. M., NORTHCUTT, J. D., AND WALL, G. A. Workstation support for time-critical applications. In *Proceedings of the Second International Workshop on Network and Operating Ssytems Support for Digital Audio and Video* (Heidelberg, Nov. 1991).
18. JEFFAY, K. The real-time producer/consumer paradigm: Towards verifiable real-time computations. Ph.D. thesis, Dept. of Computer Science, Univ. of Washington. Tech. Rep. 89-09-15.
19. KARP, R. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (Eds.) Plenum Press, New York, 1972, 85–103.
20. KUROSE, J. F., AND SIMHA, R. A microeconomic approach to optimal file allocation. In *Proceedings of the 6th International Conference on Distributed Computing Systems* (Cambridge, Mass., May 19–23, 1986).
21. KUROSE, J. F., SCHWARTZ, M., AND YEMINI, Y. Multiple-access protocols and time-constrained communication. *ACM Comput. Surv. 16*, 1, 43–70.
22. KUROSE, J. F., SCHWARTZ, M., AND YEMINI, Y. A microeconomic approach to optimization of channel access policies in multiaccess networks. In *Proceedings of the 5th International Conference on Distributed Computing Systems* (May 1985), 70–80.
23. LEINBAUGH, D. W., AND YAMINI, M. Guaranteed response time in a distributed hard real-time environment. *IEEE Trans. Softw. Eng. 12*, 12 (Dec. 1986), 1139–1144.
24. LITTLE, T. D. C., AND GHAFOOR, A. Synchronization and storage models for multimedia objects. *IEEE J. Selected Areas Commun. 8*, 3 (Apr. 1990), 413–427.
25. LIU, C. L., AND LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20*, 1 (1973), 47–61.
26. LOY, G. Designing an operating environment for a realtime performance processing system. In *Proceedings of the 1985 Inteernational Computer Music Conference* (Burnaby, B.C., Canada, 1985), 9–13.
27. LUTHER, A. C. *Digital Video in the PC Environment*. McGraw-Hill, New York, 1989.
28. MA, P., LEE, E., AND TSUCHIYA, M. A task allocation model for distributed computing systems. *IEEE Trans. Comput. 31*, 1 (Jan. 1982), 41–47.
29. MALONE, T. W., FIKES, R. E., GRANT, K. R., AND HOWARD, M. T. *The Ecology of Computation*. North Holland, Amsterdam, 1988.
30. MCAULEY, A. J. Reliable broadband communications using a burst erasure correcting code. In *Proceedings of ACM SIGCOMM 90* (Philadelphia Pa., Sept. 1990), 287–306.
31. MERCER, C. W., ISHIKAWA, Y., AND TOKUDA, H. Distributed Hartstone: A distributed real-time benchmark suite. In *Proceedings of the 10th International Conference on Distributed Computing Systems* (Paris, May 1990), 70–77.
32. MERCER, C. W., AND TOKUDA, H. An evaluation of priority consistency in protocol architectures. In *Proceedings of the IEEE Conference on Local Area Networks* (Oct. 1991).
33. MOORER, J. A. Hard-disk recording and editing of digital audio. In *Proceedings of the 89th Convention of the Audio Engineering Society* (Los Angeles, Sept. 21–25, 1990).
34. NICOLAU, C. An architecture for real-time multimedia communication systems. *IEEE Selected Areas Commun. 8*, 3 (Apr. 1990), 391–400.
35. OHTA, N., NOMURA, M., AND FUJII, T. Variable rate video encoding using motion-compensated DCT for asynchronous transfer mode networks. In *IEEE International Conference on Communications* (1988).
36. PARULKAR, G. M., AND TURNER, J. S. Towards a framework for high-speed communication in a heterogeneous networking environment. *IEEE Network 4*, 2 (Mar. 1990), 19–27.
37. RANGAN, P. V., AND VIN, H. M. Designing file systems for digital audio and video. In *Proceedings of the 13th ACM Symposium on Operating System Principles* (Pacific Grove, Calif., Oct. 1991), 81–94.
38. ROSS, F. E. An overview of FDDI: The fiber distributed data interface. *IEEE J. Selected Areas Commun. 7*, 7 (Sept. 1989).
39. SEVCIK, K. C., AND JOHNSON, M. J. Cycle time properties of the FDDI token ring protocol. *IEEE Trans. Softw. Eng. 13*, 3 (Mar. 1987), 376–385.
40. SHA, L., RAJKUMAR, R., AND LEHOCZKY, J. P. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput. 39*, 9 (Sept. 1990), 1175–1185.

41. SHEPHERD, D., AND SALMONY, M. Extending OSI to support synchronization required by multimedia applications. *Comput. Commun. 13*, 7 (Sept. 1990), 399–406.
42. SHIMADA, S., NAKAGAWA, K., AND TAKESHI, I. Gigabit/s optical fiber transmission systems—Today and tomorrow. *IEEE International Conference on Communication* (June 1986), 1538–1542.
43. WRIGHT, D. J., AND TO, M. Telecommunications applications of the 1990s and their transport requirements. *IEEE Network 4*, 2 (Mar. 1990), 34–40.
44. ZHANG, L. VirtualClock: A new traffic control algorithm for packet-switched networks. *ACM Trans. Comput. Syst. 9*, 2 (May 1991), 101–124.
45. ZHAO, W., RAMAMRITHAM, K., AND STANKOVIC, J. A. Preemptive scheduling under time and resource constraints, *IEEE Trans. Comput. 36*, 8 (Aug. 1987), 949–960.

Received November 1990; revised May 1992; accepted August 1992