META-SCHEDULING
FOR DISTRIBUTED CONTINUOUS MEDIA

David P. Anderson

Computer Science Division, EECS Department
University of California at Berkeley
Berkeley, CA 94720

September 20, 1991

ABSTRACT

Next-generation distributed systems will support *continuous media* (digital audio and video) in the same hardware/software framework as other data. Many applications that use continuous media (CM) have end-to-end performance requirements such as minimum throughput or maximum delay. To reliably support these requirements, system components such as CPU schedulers, networks, and file systems must offer realtime semantics. A *meta-scheduler* 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 meta-scheduler. The model defines a workload parameterization, an abstract interface to resources, and an end-to-end algorithm for negotiated reservation of multiple resources; the division of delay is based on an economic model. Clients make reservations for worst-case workload, and resources offer hard delay bounds. However, system components may ‘‘work ahead’’ within limits, increasing the responsiveness of bursty non-realtime workload.

1. INTRODUCTION

Much recent work has studied the use of audio and video as input/output media for user interfaces [8, 13, 31-33, 38]. We call audio and video *continuous media* (CM) because they are perceived as changing continuously over time. We say that a distributed computer system provides *integrated digital continuous media* (IDCM) if it has the following properties:

- CM data is stored and transmitted exclusively in digital form.
- Except for user-interface I/O devices, CM data is handled by the same hardware (CPU, memory, networks, I/O system) as other data.
- CM data is handled in the same software framework (programming language, operating system, file system, network communication, window server) as other data.

A comparison between IDCM and other approaches, and a high-level design of an IDCM system, are given in [3, 4].

The *video playback* program shown in Figure 1 is an example of an IDCM application. The application, running on a workstation, allows the user to select a video program to be viewed (the method of selection might involve a menu-based browser or a database query). A CM-capable window server provides user I/O [6]. When the selection is made, the application instructs the window server to prepare a ‘‘video window’’. The application then begins reading data from a remote file system and forwarding it (via local IPC) to the window server. The window server passes the data to a DSP device for decompression, causing the data to be displayed in the video window.

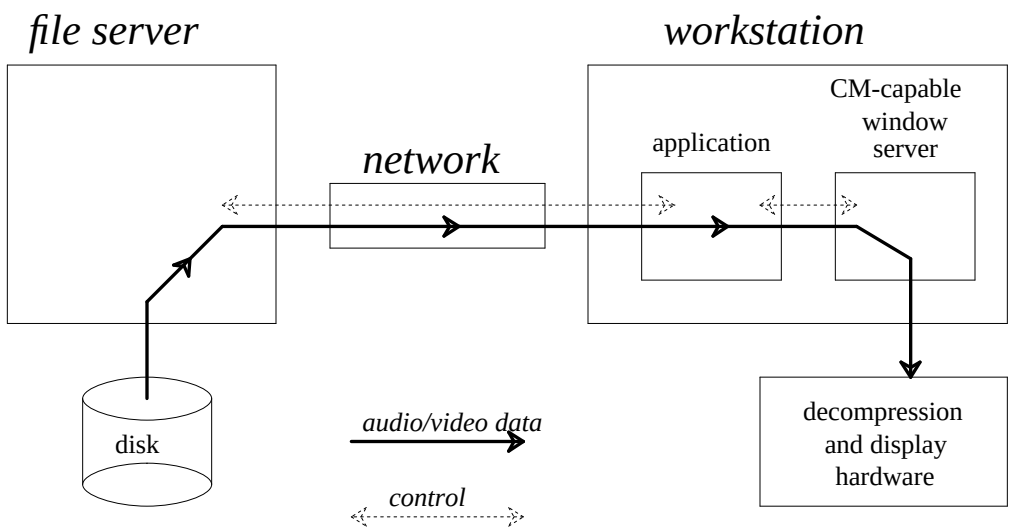

Figure 1: Video playback, a simple IDCM application. The application reads digital audio/video data from a remote file server, then sends it to a CM-capable window server.

Applications in an IDCM system may have realtime performance requirements such as bounds on end-to-end throughput and delay. For example, the video playback application requires that data be read from the disk file and transferred to the video decompression and display unit at a minimum average data rate. This rate is determined by the data representation; for example, stereo CD-quality audio is 1.4 Mbps and DVI compressed video is 1.2 Mbps [24]. Applications may also require bounds on end-to-end delay. For simple telephony, this bound is typically in the range of 100-200 milliseconds. For applications such as musical telephony it may be 10 milliseconds or less [23].

The task of an IDCM application is simplified if it can be ‘‘guaranteed’’ that the system will handle CM data with the necessary performance levels for the duration of its execution. In order to make such a guarantee, the shared components, such as CPU, file system, and network, must support ‘‘reservations’’ in which the client specifies its workload and the component provides a performance guarantee. Furthermore, to provide end-to-end guarantees it may be necessary to use a *meta-scheduler* to coordinate components (see Figure 2). A meta-scheduler is a distributed software layer that ‘‘reserves’’ components on behalf of client applications; it is not involved in the actual usage of the components.

A meta-scheduler must define a uniform model of how the components operate and interact. Some goals for a meta-scheduler and its model include:

Abstraction: Rather than prescribe a particular scheduling policy, the model should define an abstract interface that can be implemented by a range of

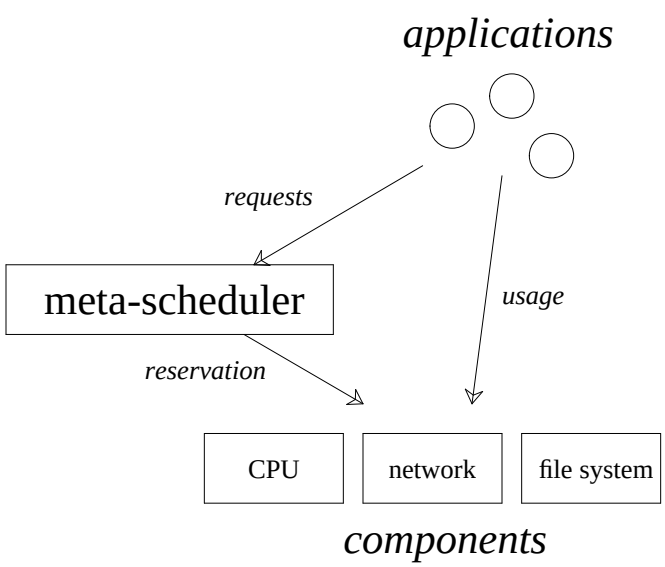

Figure 2: A *meta-scheduler* acts as a mediator between applications and realtime system components. It reserves capacity on behalf of clients; the clients then access the components directly.

scheduling algorithms, both preemptive and nonpreemptive. In this way, existing hardware and software components with realtime properties can be used with little or no modification.

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

End-to-end guarantees: To support end-to-end CM application requirements, the model must encompass all system components: CPU scheduling, I/O, networks, and so on.

The *CM-resource model* is intended as a basis for a CM meta-scheduler. The components of the model are as follows. A *resource* is a subsystem that stores, manipulates, or communicates CM data (it may handle other workload as well). Resources may be reserved in *sessions* with workload, delay, and cost parameters. Sessions may be combined into *compound sessions*. The division of delay among sessions uses an economic approach: delay is apportioned in a way that minimizes total cost. The model allows buffer space requirements to be computed, so that packet loss due to buffer overrun can be avoided.

The CM-resource model uses the *linear bounded* traffic model proposed by Cruz [10]. The model is based on conservative assumptions: clients must reserve resources based on their worst-case needs, and resources must offer hard upper bounds on delay. However, the model provides flexibility by allowing resources to ‘‘work ahead’’, making full resource capacity available to bursty non-realtime clients. In addition, any unused portion of a reservation can be used for other purposes.

CM applications may also have synchronization requirements, *e.g.*, that two streams must be displayed simultaneously or that one must immediately follow another [14, 21]. We do not discuss synchronization in this paper. However, the CM-resource model provides a basis for synchronization as well as performance guarantees.

The CM-resource model was first presented by Andrews [7]. This paper explores Andrews’ original model and offers some refinements and extensions. The paper is organized as follows. Section 2 defines the resource and session abstractions, and Section 3 deals with compound sessions. Section 4 describes techniques for implementing a CPU scheduler, an FDDI network, and a file system as resources. Section 5 discusses related work, and Section 6 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 system such as a network. Resources provide a standard interface, described in this section, for their reservation and use.

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 an interface I in the time interval $[t_0, t_1)$, where $t_0 < t_1$.

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

Definition. A *linear bounded arrival process* (LBAP) is a message arrival process at an interface I with three fixed parameters:

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

that, for all $t_0 < t_1$, satisfies

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

The long-term data rate of an LBAP is MR bytes per second. The parameter W allows short-term violations of this rate constraint, modeling programs and devices that generate ‘‘bursts’’ of messages that would otherwise exceed the constraint. These bursts consist of messages that have arrived ‘‘ahead of schedule’’; they do *not* reflect burstiness in the underlying data stream (see Section 2.8). The extent to which arrivals are ahead of schedule is quantified as follows:

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

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

We observe without proof that $w(t)$ is finite and right-continuous, and that the expression $(N(t_0, t) - R\,|t - t_0|)$ is maximized by a particular value of t_0. Intuitively, $w(t)$ is the largest ‘‘message excess’’ (relative to R) during any time interval ending at t. More concretely, $w(t)$ is a function that increases by 1 on each message arrival, decreases with slope $-R$ otherwise, and remains nonnegative (see Figure 3).

Claim 1. *$w(t) \le W$ for all t.*

Proof. The claim follows from Eq. 1 and Eq. 2. □

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

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

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

Proof. from Eq. 2, let t_0 be such that

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

Then

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

from which the claim follows. □

2.2. Describing Delay

The second central issue in our model is how to parameterize the delay between two interfaces in the system. We use a notion of delay that takes workahead 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.

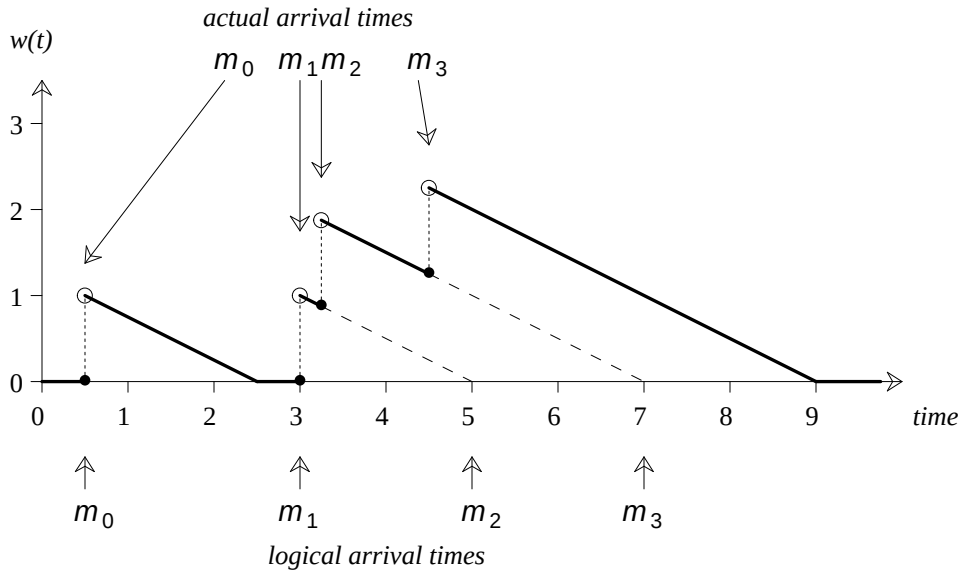

Figure 3: The workahead function $w(t)$ for an LBAP with $R = 0.5$ and message arrivals at times 0.5, 3.0, 3.25, and 4.5. The corresponding logical arrival times are 0.5, 3.0, 5.0, and 7.0.

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:

$$l(m_0) = a_0 \tag{3}$$

$$l(m_{i+1}) = \max(a_{i+1}, l(m_i) + 1/R)$$

Intuitively, $l(m)$ is the earliest time message m could have arrived if workahead were not allowed (see Figure 3; note that the logical arrival times of consecutive messages 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 motivation for this definition is as follows: If a message arrives ahead of schedule at a device and is queued there, the delay should be ''charged'' to the previous device up until the logical arrival time of the message. 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 non-negative.

2.3. Resources and Sessions

As indicated earlier, a *resource* is an entity that handles streams of CM messages (as perhaps non-realtime 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 system of interacting devices such as a network. Examples are given in Section 4.

Prior to using a resource, a client must create a *session* with the resource. Each message handled by a resource is associated with a particular session. A session has 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)
U = minimum unbuffered actual delay (seconds)

The client must ensure that the arrival process at the input interface obeys the LBAP parameters M, R and W_{in}. The resource must ensure that the arrival process at the output interface obeys the LBAP parameters M, R and W_{out}. D is an upper bound on the logical delay, between the input and output interfaces of the resource, of any message associated with the session. A is a lower bound on the actual delay. U is a lower bound on the actual delay of a message during which it is not stored in host memory (see Section 3.3).

In general, if a resource does queueing, the outgoing workahead may be larger than the incoming workahead. However, it is possible for a resource to reduce its output workahead by doing *regulation*, *i.e.* by delaying outgoing messages that are completed ahead of schedule (see Section 4.3).

A session is an agreement between the client and the resource. The resource ''guarantees'' that it will obey the delay bounds and output workahead limit. Like any guarantee, this does not hold with probability one. For a given session, there is a nonzero probability that a guarantee will be violated in a given time interval (due, *e.g.*, to a hardware failure). In the CM-resource model we assume that this probability is low enough so that it can be treated as an anomaly. Clients are not informed of the probability, and we do not specify if or how clients are informed when a guarantee is violated.

Likewise, the client guarantees that it will not exceed the workload parameters. We do not specify whether a resource checks for compliance with the LBAP parameters, or how it responds to violations. The simplest assumption is that the resource trusts the client to obey the parameters, so that no checking is needed.

If the client workload is ''bursty'' (due to a variable data rate encoding, silence suppression, *etc.*) then the client must make a reservation based on the maximum data rate. A session is a ''reservation'' of part of a resource in the sense that, if workload is presented according to the session parameters, it must be handled within the logical delay bound. However, unused resource capacity can potentially be used for other purposes if the workload does not arrive at the maximum rate.

There is no requirement for clock synchronization or a global time source, even if a resource (such as a network) spans multiple hosts. The workahead function, which determines local scheduling deadlines, 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.

2.4. Non-Starvation of Output Devices

Let us now return to the video-playback example of Section 1. Suppose that the audio/video output hardware requires a packet of data every .05 second. If data fails to arrive on time, the device is momentarily ''starved'', producing a noticeable pause in the output. This starvation may occur even if data is read from the disk at the proper rate, as shown in the following scenario. Suppose the network session has a delay bound of 1 second, and that a packet is read from the disk and submitted to the network every .05 second. Suppose packet 0 is read from the disk at time $t = 0$, and traverses the network in 0.1 second. The receiver immediately delivers it to the video device at $t = 0.1$. Packet 1 takes 1 second to traverse the network, arriving at the receiver at $t = 1.05$. The device is starved between $t = 0.1$ and $t = 1.05$.

We will now show that the receiver can eliminate this starvation by delaying the start of output.

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

Claim 3. *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. □

Definition. Suppose a client obtains data from the output interface of a session with delay bound D and sends it to a device that requires data at periodic intervals. The client is said to be *conservative* if it waits until time $t_1 = a_0 + D$ to start outputting the data.

Claim 4. *Suppose that the arrival process of a session is workahead-positive and that the receiver is conservative. Then the receiver never starves.*

Proof. Assume otherwise. Then there is some 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 4). Then $w(t_3) > 0$, so by Eq. 4, $N(t_0, t_3) > R\,|t_3 - t_0|$. But all messages arriving between t_0 and t_3 must arrive between t_0 and t_2, and $t_3 - t_0 = t_2 - t_1$, contradicting Eq. 5. □

Actually, a 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.

2.5. Delay-Cost Functions

For a given throughput, a session with a smaller delay bound is more ''costly'' to a resource because it limits the resource's ability to accommodate other small-delay

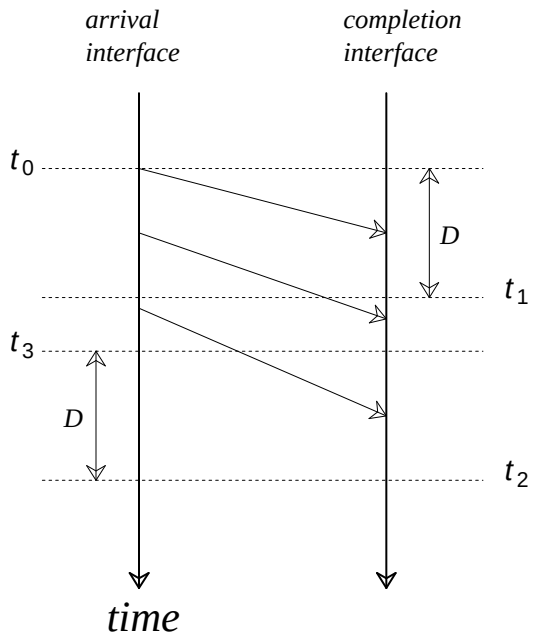

Figure 4: Diagram for the proof of Claim 4.

sessions. In the CM-resource model this cost is quantified and is used to determine delay division between resources. The establishment of a session consists of three steps:

(1) The client requests a session, giving the session's message size and rate.

(2) If it can accept the session, the resource returns the minimum possible logical delay bound $D_{\min}$ for the session, and makes a resource reservation sufficient to provide this bound. It also returns a ''cost function'' $C(D)$ whose value at D is the cost per unit time of maintaining a session with throughput R and delay bound D.

(3) Based on the cost function C, the client decides on a specific delay bound $\bar{D} \geq D_{\min}$, and tells the resource to ''relax'' the existing reservation by changing the delay bound to $\bar{D}$.

Cost functions are used in dividing delay among a sequence of resources in a way that minimizes total cost. To ensure that the cost minimization problem is tractable (see Section 3.5), we require that cost functions have the following property:

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

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

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

A tractable cost function is defined on an interval $[d_1, d_2]$. d_1 is the smallest delay

bound that the resource can currently provide. Beyond d_2, the ‘‘cost’’ of buffering messages over the delay period exceeds the cost saved by the larger delay.

2.6. The Session Reservation Interface

Each resource has an associated software module that exports the following interface:

```
reserve()
   input parameters:
      maximum message size and rate
   output parameters:
      success/failure flag
      session ID
      delay-cost function
      minimum actual delay
      minimum unbuffered actual delay

relax()
   input parameters:
      session ID
      new maximum logical delay

free()
   input parameter:
         session ID
```

`Reserve()` requests a session with the given workload parameters and the smallest possible logical delay bound. `Relax()` increases the delay bound of an existing session. `Free()` deletes an existing session. The details of the interface may vary between resource types. For example, a CPU resource’s `reserve()` operation would have a ‘‘maximum CPU time per message’’ parameter instead of the maximum message size, and a network resource would be given the destination host address.

Note that the interfaces do not refer to workahead limits. We make the simplifying assumption that resources on a host share a common pool of buffer memory, and therefore workahead limits are relevant only at the interface between hosts (*i.e.*, network resources). This will be explained further in Section 3.

2.7. Source and Sink Resources

The resources we have considered so far are those that *handle* CM data, either by processing it or by transporting it. Other types of devices 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 devices as providing a single input or output interface, with associated LBAP parameters. There is no notion of delay, so the interface exported by such resources is a simplified version of the one shown above.

2.8. Assumptions, Rationale, and Refinements

We now give the assumptions and rationale behind the interrelated basic components of the CM-resource model: arrival rate, delay, and loss. We then describe some possible refinements to the model.

2.8.1. Arrival Process Model

In discussing distributed CM systems, it is important to distinguish between 1) the model of the underlying data stream (CM data being produced or consumed by a CM I/O device) and 2) the model of the arrival at interfaces within the system. These models may be different if data is delayed by variable amounts within the system. Several models for the underlying data stream are relevant to CM:

(1) The data rate is constant.

(2) The data rate is constant over ‘‘talk bursts’’ of a few seconds, with interspersed ‘‘silences’’. This is a good model for voice conversations [9].

(3) The data rate is dynamically selected from a set of discrete values. For example, if available bandwidth declines, high-frequency information can be omitted, with a resultant loss in image or sound quality.

(4) The data rate varies dynamically. Such ‘‘burstiness’’ may be described by the mean-to-variance ratio or the peak-to-average data rate ratio. Some video compression schemes result in a variable data rate [28].

The CM-resource model requires that a data stream have a bounded data rate, and it requires clients to make reservations based on this rate. Hence it can express all four data models, although in cases 2, 3, and 4 part of the reserved capacity will not be used by the session. We chose not to model bursty underlying data for several reasons:

- Models that allow resources to be ‘‘overbooked’’ (based on underlying data with random burstiness) force applications to deal with packet loss and/or unbounded delays.
- Models in which applications must vary their underlying data rates according to changing system loads (*e.g.*, model (3)) increase the complexity of all system components, and are justified only when hardware resources are scarce.
- If the CM-resource model is used for a bursty underlying data stream (models (2) and (4)), the unused resource capacity is not wasted: it is unavailable for other reservations, but can be used by non-realtime traffic or by workahead realtime traffic. If hardware resources are sufficient so that reservations are not normally turned down, nothing is lost by over-reservation.
- Some CM data representations are not bursty. These include uncompressed digital music (in which there are no silences) and video compression schemes such as DVI.

The rationale for using the LBAP model for intra-system interfaces is as follows: Given the bounded-rate model for underlying data and the goal of eliminating buffer overrun (see below), the LBAP model is in a sense the least restrictive possible model. Its workahead limit parameter corresponds to a bound on buffer space, and it makes no other demands (*e.g.*, minimum inter-message gap) on the workload.

2.8.2. Loss Model

Message loss in a distributed system can occur because of data corruption or buffer overrun. For CM applications, output data that arrives late or out of order is lost in the sense that it cannot be used. This type of loss cannot be remedied by transport protocols that do retransmission.

The simplest loss model is the zero-loss model: errors and losses are as improbable as memory or processor errors. This is the assumption and goal of the CM-resource model. Future networks, especially those using fiber-optic transmission media,

will have a bit error rate as low as the processors in the end nodes. Therefore, if the system can eliminate losses due to other sources, the distributed system can be viewed as error-free, greatly simplifying applications.

A contrasting assumption is that CM data is error-tolerant because ‘‘glitches’’ are transient, and models based on this approach include error-rate parameters. However, the assumption does not always hold. Data representations such as differentially compressed video are error-intolerant, glitches in CD-quality audio are often unacceptable, and some applications (such as archival recording) require near-zero error rate.

2.8.3. Delay Model

The delay introduced by a device or subsystem can be modelled in several ways. Examples include:

- **Probabilistic bounds:** A resource might provide an upper bound on the 90th percentile delay (*i.e.*, 90% of messages are delayed less than *X*).
- **Backlog-avoiding bounds:** The delay within a resource is bounded by the message interarrival time (this assumes approximately periodic arrivals). This model has been explored in the context of traditional real-time systems [22].

We rejected probabilistic approaches because they imply a non-zero loss rate for CM output applications. The CM-resource model requires an upper bound on logical delay, but is more flexible than backlog-avoiding models in two ways: 1) the bound can be greater than the average interarrival time; 2) 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 non-realtime traffic, as shown in Figure 5. Context switching overhead is also lower.

A meta-scheduling model must adopt some criterion for dividing delay between resources. One approach, based on work in network congestion control [15], is to explicitly use load metrics. We chose to use cost as a basis because of its universal and absolute nature: commercial data networks charge real money based on quality-of-service parameters. The cost functions of resources will typically be coupled to their load; 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.

2.8.4. Refinements For Variable Data Rate

The model as described has no notion of data ‘‘timestamps’’, and this can result in suboptimal scheduling decisions if the underlying data stream is variable-rate. For example, suppose a session has a maximum rate of 10 messages per second, but in the first 10 seconds the data stream uses only 1 message per second. If the data source works ahead and generates 10 messages immediately, the 11th message will be assigned logical arrival time 1 and will be scheduled accordingly. In fact, however, it should be given logical arrival time 10, and therefore given lower priority.

This problem can be solved by including timestamps (or time differences, from which timestamps can be computed) with messages. These timestamps can be used instead of logical arrival time in determining scheduling priorities.

3. COMPOUND SESSIONS

A meta-scheduler must address, at the minimum, situations in which data traverses a linear sequence of resources. For example, in the video playback application of Section 1, data originates from a disk, traverses a CPU, a network, and another CPU, and is

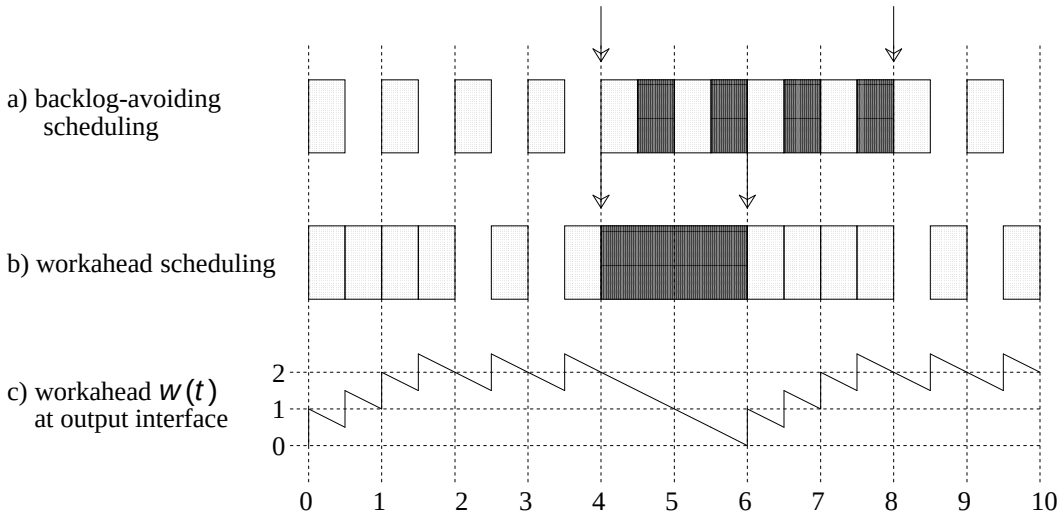

Figure 5: A comparison of workahead and backlog-avoiding scheduling. A CPU resource is handling a stream of CM data (light boxes) that uses 50% of the CPU capacity. A 2-second non-realtime task (dark boxes) arrives at time 4. Workahead scheduling is 2 seconds ahead of schedule at this point (c) and can handle the new task in in a single uninterrupted CPU burst (b). With backlog-avoiding scheduling (a), the new task is timesliced and is not finished until time 8.

then consumed by a video decompression chip. In the CM-resource model, such a situation is represented as a ''compound session'' consisting of sessions with each of the resources involved. The following issues must be addressed:

- How can a given bound on end-to-end delay be divided ''fairly'' among resources? (It may not be fair, or even possible, to divide it uniformly.)
- How much buffer space must be reserved in each host to avoid packet loss due to buffer overrun?
- 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.

3.1. Terminology

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} (see Figure 6).

The resources in a compound session handle a stream of messages in pipeline fashion. All the sessions must therefore have the same throughput limit. Furthermore, the output workahead limit of session S_i cannot exceed the input workahead limit of session S_{i+1}. The arrival process of S is defined as the arrival process of S_1; similarly the completion process of S is the completion process of S_n. The logical delay d of a mes-

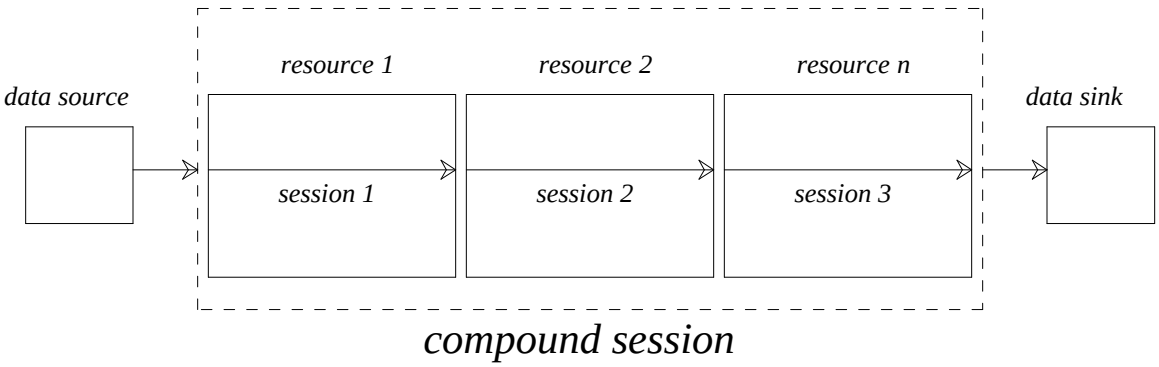

Figure 6: A compound session consists of a sequence of sessions in which output interface of S_i is the input interface of S_{i+1}.

sage m in S is the difference in logical arrival time of m at these two interfaces.

Claim 5. *Let d and S be as above. Then $d = \sum d_i$ where d_i is the logical delay in 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

$$\begin{aligned} d &= (l_n - l_0) \\ &= (l_n - l_{n-1}) + (l_{n-1} - l_{n-2}) + \cdots + (l_1 - l_0) \\ &= d_n + \cdots + d_0 \end{aligned}$$

□

Claim 6. *Let d and S be as above. Then $d \leq \sum_{i=1}^{n} D_i$, where D_i is the maximum logical delay in S_i.*

Proof. The claim follows from Claim 5. □

3.2. Non-Starvation in Compound Sessions

The non-starvation result of Section 2.4 can be extended to compound sessions. Consider a situation (such as the video-playback application of Section 1) in which a ‘‘receiver’’ process takes data from the output interface of a compound session S and sends it to an output device that requires periodic arrivals.

Claim 7. *Suppose that the arrival process of a compound session is backlog-maintaining and that the receiver does conservative output (i.e., it delays output until $t_0 + D$, where t_0 is the arrival time of the first message at S, and D is the logical delay bound of S). Then the receiver will never starve.*

Proof. The claim follows from Claims 6 and 4. □

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

The extra buffer space required for conservative output (beyond that needed to accommodate incoming workahead and delay, to be discussed in Section 3.3) is εR in case (1), zero in case (2), and $R(D - A_{\min})$ in case (3).

3.3. Buffer Space Requirements of a Compound Session

In this section we compute a bound on the buffer space in a given host required to prevent buffer overrun for a compound session. Consider a compound session S that traverses a host H. Let $R_1 \cdots R_n$ be the set of resources in S for which messages are buffered in the main memory of H. If S includes a network resource moving data into H, this resource is not included in the list R, since the output interface of the network resource is the moment of message arrival in host memory (see Section 4). R_n may be a network (see Figure 7), in which case its minimum unbuffered time U (corresponding to propagation time) is nonzero. Each resource has a logical delay bound D_i; let $D = \sum_i D_i$. Let W be the incoming workahead limit of R_1.

Claim 8. *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 oldest message in host memory. Let a be its arrival time at R_1, and $w(a)$ be the workahead at R_1's input interface at time a. The number of messages buffered in the host at time t is $N(a, t)$, the number of arrivals to R_1 in $[a, t)$ (see Figure 8). From Claim 2 we have

$$N(a, t) \le w(t) - w(a) + R|t - a| \tag{6}$$

$$\le W - w(a) + R|t - a|$$

since $w(t) \le W$. Now let b denote the actual completion time of m in R_n.

$$D \ge l(m)$$

$$= (b + w(b)/R) - (a + w(a)/R)$$

$$\ge b - (a + w(a)/R)$$

so

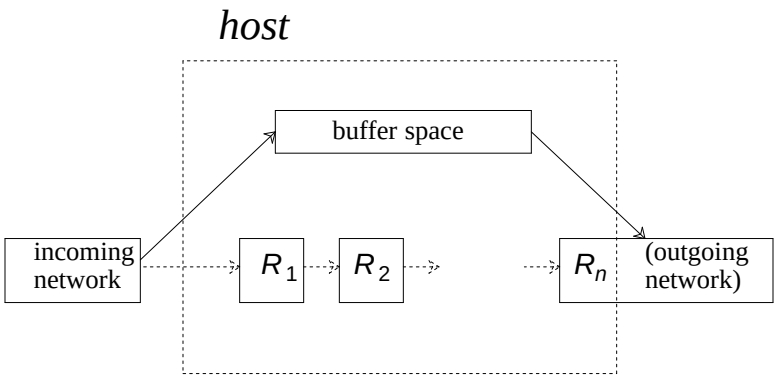

Figure 7: A compound session can involve several resources in a given host. Except for the outgoing network, messages are buffered in host memory while they are being handled by these resources. In the outgoing network, messages are buffered only for part of the time.

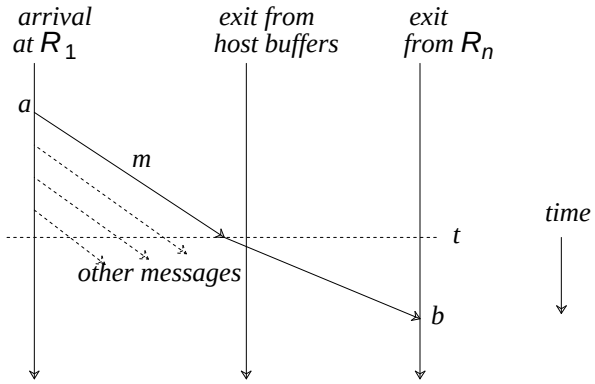

Figure 8: Diagram for the proof of Claim 8. The number of messages buffered at time t is $N(a, t)$, where a is the arrival time of the oldest message in memory at time t.

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

Now from the definition of U,

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

Combining Eq. 7 and Eq. 8 we have

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

Now combining equations Eq. 6 and Eq. 9 we have

$$N(a, t) \le W - w(a) + R(D + w(a)/R - U)$$

$$\le W + R(D - U) \quad (10)$$

□

The above 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.

The receiving host may also do buffering as a consequence of conservative output. In this case the additional buffer space is required, as described in Section 3.2.

3.4. Workahead Limits in Compound Sessions

Resources may do regulation to reduce their outgoing workahead. This may be necessary in compound sessions, since otherwise buffer space requirements could grow without bound [10]. Under the assumption that resources on a host share buffer space, the total host buffer space requirement depends only on the workahead into R_1, 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 reservation interface of network resources must be augmented as follows. The `reserve()` operation 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.

3.5. Compound Cost Functions

The cost per unit time 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 $\bar{D}$, find resource delay bounds $D_1 \cdots D_n$ that solve

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

subject to

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

Claim 9. *If the C_i are arbitrary piecewise-linear functions, then the minimal-cost delay problem is NP-hard.*

Proof. We show this by reducing the NP-complete PARTITION problem [16] 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 $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$? Consider the following instance of the minimal-cost delay problem: for each $a \in A$, let C_a be the function

$$C_a(d) = s(a), \ d < s(a)$$

$$= 0, \ d \ge s(a)$$

and let $\bar{D} = \frac{1}{2}(\sum_{a \in A} s(a))$. 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 $C = \sum_{a \in A} C_a(d(a))$. Then $C = \bar{D}$ iff there is a solution to the original PARTITION problem. □

If cost functions are tractable (Section 2.5) 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 9).

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

(2) Position the first (steepest) segment so that it starts at the vector sum of the left endpoints of the functions.

(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 measure of the set

$\{x \leq d : \langle x, C(x) \rangle$ is labeled with $R\}$

(see Figure 10).

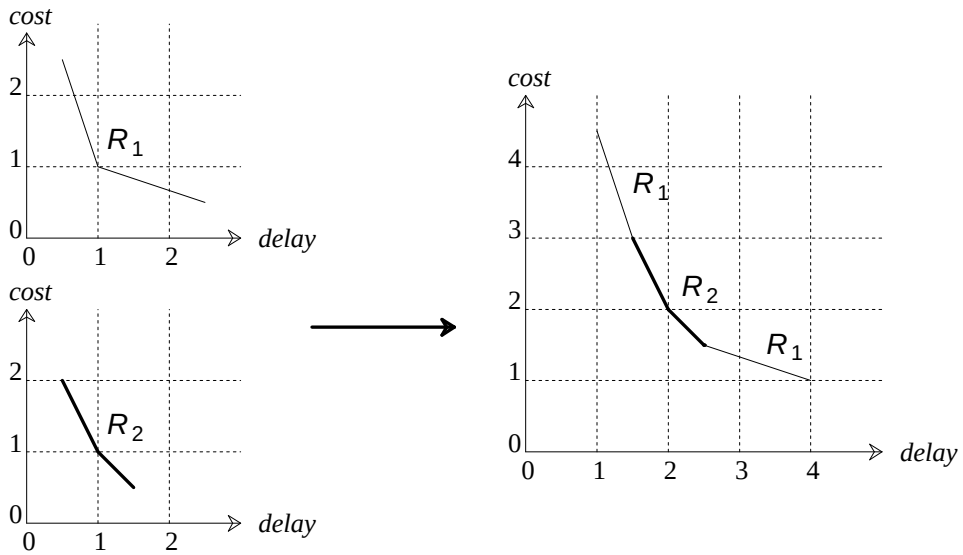

Figure 9: The cost functions for several resources can be combined to form a *compound cost function* whose segments are labeled with the names of the resources.

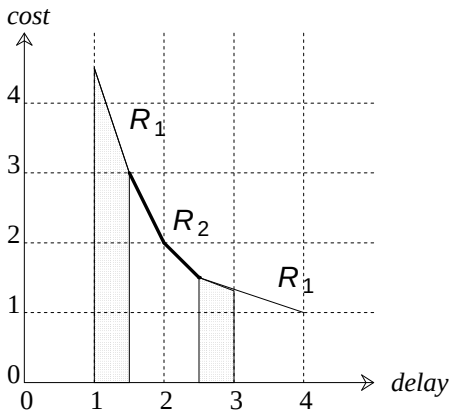

Figure 10: The function $F(C, R, d)$ returns the optimal amount by which to relax the reservation of resource R given an end-to-end delay d. In this example, $F(C, R_1, 3) = 1$.

Claim 10. *The delay assignment given by $D_i = F(C, R, D)$ solves the minimal-cost delay assignment problem.*

We omit the proof for brevity.

3.6. The Compound Session Establishment Protocol

The CM-resource model defines a protocol for establishing compound sessions. The protocol approximates the minimum-cost division of delay subject to host buffer space limits. The protocol involves an interaction between sending and receiving clients, resources, *host resource managers* (HRMs), and buffer space managers on each host (see Figure 11). We assume that the memory manager has operations

`reserve_memory(n)`: reserve *n* bytes of physical memory.

`n = reserve_extra_memory()`: reserve additional physical memory for a compound session. The amount of memory reserved is determined by a host-specific policy.

`free_memory(n)`: free an existing reservation of *n* bytes.

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.

- 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 in no case greater than $E_{\max}$; and, given this delay bound, to minimize the cost of the session.

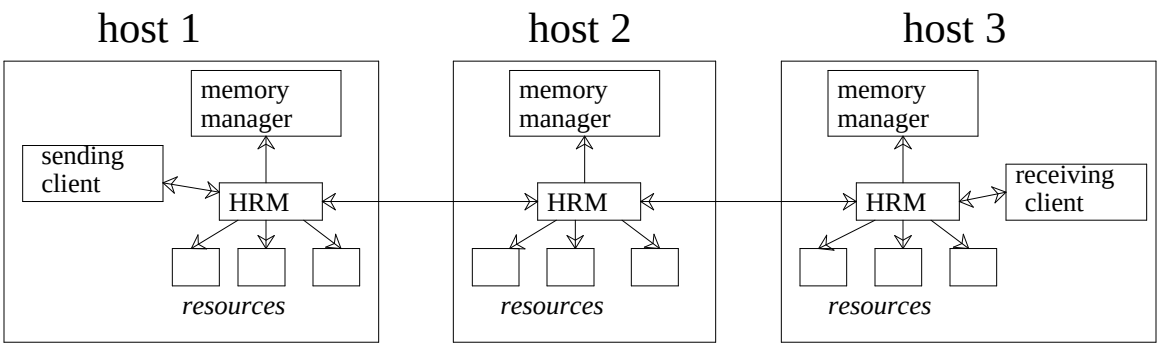

Figure 11: The participants in the compound session establishment protocol include clients, host resource managers (HRMs), memory managers, and resources.

- 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 a compound session establishment by passing a request message to its local HRM. For the time being, we can assume that C is empty and D is zero (for other cases see Section 3.7). Each HRM executes the following algorithm. Suppose the local resources involved in the compound session are $R_1 \cdots R_n$.

(1) Do the `reserve()` operation on $R_1 \cdots R_n$ in any order or in parallel.

(2) Call `reserve_memory()` to get buffer space according to Eq. 10.

(3) If any of the above operations fails, or if the total delay exceeds $E_{\max}$, free all reservations and returns a failure message.

(4) Prepare an outgoing request message. D, A, and C are updated according to the results of the `reserve()` operations. The new outgoing workahead W is returned by the `reserve()` operation on R_n (the network resource). 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, perhaps based on C. If it does conservative output, the receiver may need to reserve additional buffer space (see Section 3.2). The *excess logical delay* E_{excess} is computed as

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

Phase two then 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. Each HRM executes the following algorithm.

(1) Call `reserve_extra_memory()` to allocate additional buffer space; 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. Do the `relax()` operation on R_i, increasing its delay bound by x. Subtract x from E_{excess}.

(3) Compute the largest value of W_l such that total buffer space needs do not exceed B. Prepare a new reply message, containing W_l 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.

This algorithm 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.

Resources are reserved for the minimum possible delay bound in phase one, then relaxed in phase two. A session request arriving in the interim could be rejected, even though it might be accepted after relaxation. To avoid this pathological situation, the following policy can be used. If a local reservation request fails and there are compound session requests outstanding, the request is put in a FIFO queue of requests for that resource, and retried when a `relax()` is done. If the request still 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.

3.7. Nonlinear Compound Sessions

The notion of compound session can be extended to include situations in which data streams branch or merge. Such ‘‘nonlinear compound sessions’’ can be established using the mechanisms for linear compound sessions described above. These mechanisms 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 receiving a session request from the HRM, can do whatever it wants (including establishing more sessions) before it initiates phase two.

As an example, consider an application in which a stream of CM data is sent synchronously to multiple receivers. This can be done efficiently using a *multicast tree* [11]. By making each edge of this tree a compound session, it is possible to form a *tree-structured compound session*, thus guaranteeing the throughput and delay to each of the recipients (see Figure 12).

The algorithm for establishing a tree-structured session is as follows. A tree of *multicast agents* $X_1 \cdots X_n$ is given (we do not specify how this tree is determined). The root is the data source and the leaves are 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 executes the following algorithm.

(1) Receive a session request message M from the local HRM (the root agent X_1 receives its request directly from a client).

(2) For each child X_i, asynchronously call HRM, requesting a compound session traversing the CPU and network resources to X_i. The cumulative cost functions

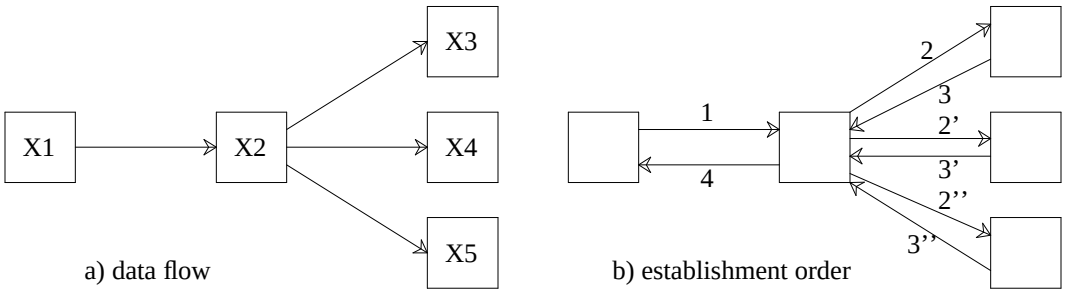

Figure 12: A simple example of a tree-structured session. Multicast agent X_1 receives CM data from X_1, and forwards it to X_3, X_4 and X_5. When establishing the session, the order of messages is shown in b); (2, 2', 2'') and (3, 3', 3'') occur in parallel.

and delays for this request are taken from M.

(3) Wait for replies from all children. Let D be the largest of the delays of the children, W be the minimum of the workaheads from the children, and E be the minimum of the excess delays.

(4) Form a reply message containing D, W and E; pass this reply message to the local HRM.

In a second example, a receiver receives data streams from several sources and uses an audio DSP to mix the streams. It can do this as follows:

(1) Wait for session requests to arrive from each of the sources.

(2) Do a `reserve()` operation on the DSP resource.

(3) Compute the excess delay for each of the component sessions; let E be the minimum of these.

(4) For each incoming session i, compute $F(C_i, R, E)$ where R is the DSP resource. Let X be the minimum of these. Call `relax()` to return this amount of delay to the DSP resource.

(5) For each incoming session i, construct a reply message containing the appropriate E_{excess} and return it to the local HRM.

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 optimal cost assignment.

3.8. Elaborations of the Compound Session Model

The compound session model might be extended in several ways:

Negotiated message size. The model can be extended so that message size, and its associated packetization delay, is negotiated by the session establishment algorithm.

Route selection. The session establishment algorithm can be extended to select between multiple alternative routes based on cost minimization.

Changing conditions. The following mechanisms could be added to deal with changing conditions: 1) Renegotiate a session, dividing the delay differently, when either resource costs change (*e.g.*, due to changing load conditions) or more buffer space becomes available. 2) If a client request is rejected due to insufficient resources, arrange to notify the client when resources are available. 3) For CM applications that change data rates, renegotiate the throughput of a session (as opposed to deleting the session and creating a new one with a different throughput).

4. IMPLEMENTING THE CM-RESOURCE MODEL

We now discuss how the CM-resource model can be put into practice. First, we describe an implementation of the compound session establishment algorithm for the DoD Internet. We then illustrate how a CPU scheduler, a token-ring network, and a file system might be implemented in the CM-resource model (*i.e.*, how they can be managed and scheduled in order to support the interface defined in Section 2.6). These examples are intended only to illustrate representative techniques, and are not detailed or complete.

4.1. SRP: An Internet Protocol for Session Establishment

The Session Reservation Protocol (SRP) is 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 traffic from the sending to the receiving client on the associated connection. No modifications to the format of transport- or IP-level messages, or to the transport protocols themselves, is required. A prototype of SRP has been implemented under Mach. It runs as a user-level server process that communicates with clients via local IPC, with other SRP servers via Sun RPC, and with kernel-level resources via special system calls (see Figure 13).

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 sending client and SRP.

SRP requires that the connection follow a static route through the network, so that the set of resources is fixed. This 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].

4.2. Uniprocessor as Resource

We assume that for each session S there is a process P_S that does all the work for S, and no other work. P_S handles a sequence of messages arriving asynchronously (say, on a network connection), and sleeps whenever no messages are available. The input interface for S is defined by message arrivals; in the case of a network connection, this is the moment the network interface requests a receive interrupt. Message

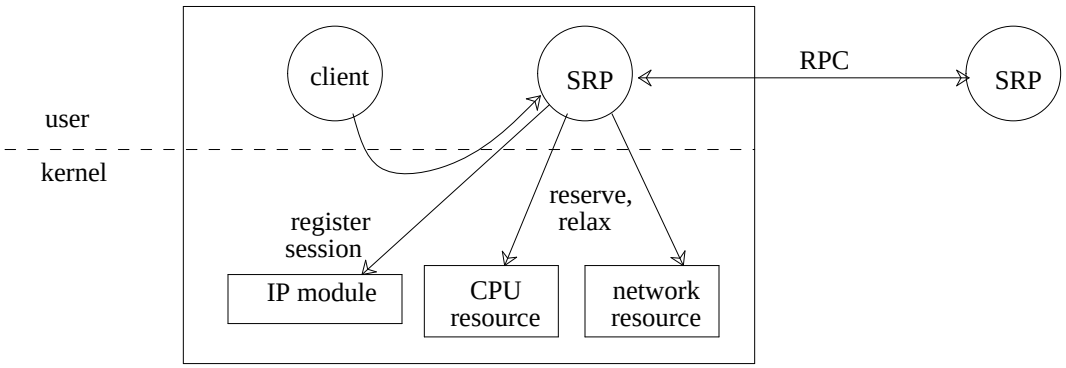

Figure 13: The Session Reservation Protocol (SRP) is implemented by a set of user-level servers that communicate with clients and with each other, and that make system calls to reserve local resources.

completion occurs when P makes a call indicating that it has handled the packet; this call either changes the priority of P or puts it to sleep if there are no more messages.

We propose a CPU scheduling policy called *deadline-workahead* scheduling in which processes are (dynamically) classified as follows. A *realtime process* is one that is associated with a session. At a given time t, a realtime 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). Realtime processes that have pending work but are not critical are called *workahead* processes. There are two classes of non-realtime processes: *interactive* (for which fast response time is important) and *background*.

The deadline-workahead policy can be summarized as follows. Critical processes have priority over all others, and are preemptively scheduled according to earliest deadline (the deadline of a process is the logical arrival time of its first unprocessed message plus its delay bound). For each workahead process, the scheduler uses a timer to make the process critical at the appropriate time. Interactive processes have priority over workahead processes, but are preempted when those processes become critical. Non-realtime processes are scheduled according to an unspecified policy, such as the UNIX time-slicing policy. This policy may also move a process between interactive and background.

When there no runnable critical or interactive processes and a workahead process is runnable, the scheduler chooses a workahead process P (perhaps the process with the earliest deadline or the most work available). P is then run for a full quantum (say, 100 times the system call plus context switch time) even if its deadline advances beyond that of another workahead process. This reduces the context switch overhead during workahead.

The `reserve()` operation exported by our CPU scheduler takes two additional input parameters: minimum and maximum CPU time per message. The scheduler maintains a list of existing sessions and their parameters. 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 is determined by iterated simulation (see [7]). Initially, D is set to the minimum CPU time per message of S. Then the following steps are done.

(1) Simulate the operation of the CPU under a ''worst-case'' workload: all sessions generate maximum periodic workload starting at the same time. If this case can be handled without missing a deadline, so can any other [22].

(2) If the simulation reaches a point where the run queue is empty, terminate; D is the delay bound of S.

(3) If a message completion at time t for a session other than S violates its delay bound, increase D to $t - t_0$, where t_0 is the last arrival of a message for S, and go to step (1). (By increasing D by this amount, the deadline will not be missed on the next simulation.)

(4) If a message completion for S violates its deadline by an amount X, add X to D and go to step (1).

In order for the scheduler to work correctly, CPU execution must be matched with the correct session and message. System software must be structured in a way that makes this possible. For example, incoming protocol handling must be done by processes, rather than by a software interrupt routine as in current BSD UNIX-based systems. When realtime 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 non-preemption), *etc.* Methods for bounding these, and taking them into account during reservation, exist [35], but are beyond the scope of this paper.

4.3. FDDI Network as Resource

FDDI is a 100 Mbps local-area network that uses a *timed token* media access protocol [39, 46]. The FDDI protocol has two data priorities. *Asynchronous* data uses a conventional token-passing protocol: each station waits for the token, transmits for a bounded time, and regenerates the token. *Synchronous* data has higher priority. A Target Token Rotation Time (TTRT) is fixed[1]. If a given station has not seen the token within the TTRT then it may not send asynchronous data. Each station i has an allotment S_i for synchronous data; it may only send that many bytes each time it gets the token. The allotments are initially zero, and can be adjusted by making requests to a network manager using a Station Management (SMT) protocol [45]. The network manager ensures that

$$P + (\sum_{hosts\ i} S_i)/B \le TTRT$$

where B is the data rate of the network and P is the propagation time around the ring. If this holds, then the maximum token rotation time is 2(TTRT), the average token rotation time is TTRT [34], and each station i is guaranteed a minimum throughput of

$$\frac{1}{TTRT} S_i \tag{11}$$

bytes per second for synchronous data.

[1] The TTRT can be a critical factor in determining the range of sessions that the network can handle. Valenzano *et al.* [40] analyze the effect of TTRT on network performance.

Let us consider how to provide CM-resource sessions in an FDDI network. A session in this case consists of a point-to-point connection between two hosts. Message arrival into a session is the moment of 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 provide delay bounds, and how to limit workahead. We discuss these separately.

4.3.1. Scheduling and Reservation

The crucial hardware resource here is the network medium[2], for which there are two levels of scheduling: 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 the following policy for 2). `Network_send()` computes the workahead of the packet and, based on this, the deadline by which it must be transmitted. The send queue is deadline-sorted. On each send-completed interrupt, the queued packet with the earliest deadline is sent.

The `reserve()` operation, like that for the CPU, uses simulation. In this case the parameters of a simulation are 1) the delay bounds of the sessions and 2) the host's synchronous-data allotment A. Each simulation models the system under worst-case conditions (all sessions begin simultaneously, 2TTRT elapses before the token arrives initially, and TTRT elapses between subsequent token arrivals). The simulator is used iteratively to solve two problems:

Delay minimization. The goal here is to find the smallest possible delay bound for a new session, and a corresponding allotment A, given an upper bound $\bar{A}$ on A. An initial delay bound D and allotment A are given. The simulation is run. If a deadline is missed during the simulation, A is increased by the smallest amount that allows a packet to be transmitted in an earlier token round, and the simulation is restarted. If such an allotment would exceed $\bar{A}$, then A is set to $\bar{A}$, the delay bound D is increased by the smallest amount that would result in a different transmission order, and the simulation is restarted.

Allotment Minimization. The goal here is to find the smallest possible allotment given fixed delay bounds. An initial allotment A is given. On each iteration, if 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 goal of `reserve()` is to make a reservation for the smallest possible delay; it can make as large an allotment as it needs (and is able) to. The algorithm is as follows: The host contacts the network manager, requesting the largest possible allotment $\bar{A}$. If this is not sufficient to support the aggregate throughput of this host's sessions by Eq. 11, the allotment is returned to its previous level and the new session is rejected. Otherwise, the simulator is used to solve the Delay Minimization problem described above. The initial delay bound for the new session is 2(TTRT) (the simulator computes only media-access delays; the actual delay bound returned by `reserve()` includes transmission time and propagation time also). 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

[2] We ignore the small amounts of CPU time needed to do queueing operations and interrupt handling on both hosts. CPU time for time-consuming activities such as data checksumming is assumed to have been reserved (with the CPU resource) by higher levels.

described below.

The goal of the `relax()` operation is to reduce the allotment as much as possible, given a new delay bound. This is done by using the simulator to solve the Allotment Minimization problem described above, using the delay bound supplied as an argument to `relax()`. The resulting allotment is communicated to the network manager. Likewise, `free()` computes the minimum allotment for the new set of sessions, and communicates it to the network manager.

4.3.2. Regulation of Packet Transmission

The transmission of network packets must be ‘‘regulated’’ to prevent receive buffer overflow. This regulation can be accomplished as follows. When a packet is passed to `network_send()`, the workahead of the session is computed according to Eq. 3. If the result exceeds the workahead limit for this session, the packet is enqueued in a separate *delayed-send queue*, and a per-session *delayed-send timer* is started if it is not already active. The delayed-send timer is set for a time when the workahead will be below the session’s limit. When the delayed-send timer for a given session expires, one or more packets from that session are sent on the relevant network interface, or are enqueued for sending.

To reduce the timer overhead, it may be desirable to use hysteresis, setting the timer so that the workahead will be below a fraction (say 50%) of the workahead limit. This avoids setting a timer for every packet sent during workahead.

4.4. The File System as Resource

We now sketch the design of CMFS, a file system that can serve as a source or sink of both CM data and non-realtime data. For simplicity, we assume that data is stored on a single disk drive, and we consider only reading. We also assume that the CPU time of interrupt handling and disk scheduling decisions is negligible.

CMFS provides a conventional interface for file creation and access. In addition, it allows clients to create *realtime files*, and exports an operation

```
read_RT_file(
      int   R,
      void* buffer,
      int buffer_length);
```

that causes a realtime file to be read at a specified rate R (bytes/second) into a circular memory buffer, without explicit read calls by the client. `Read_RT_file()` returns failure if it is not possible to read the file at the given rate. Otherwise, the client is notified when playback has begun. The arrival process at the buffer is guaranteed to be workahead-positive relative to this start time.

We assume that the only available buffer space is that supplied by clients. The reservation and scheduling algorithms used by CMFS must ensure that the rate guarantees of all sessions are honored given this limit on buffer space.

We assume that the disk is laid out so that for a given file F, there is a fixed block size B_F (in bytes) and a maximum seek/rotation time T_F between adjacent blocks[3]. There are two global parameters: maximum seek/rotation time T_G between any two blocks, and data transfer rate R_G.

[3] These are functions of the disk layout policy, which we do not discuss here.

Suppose that CMFS has active sessions $S_1 \cdots S_n$ reading files $F_1 \cdots F_n$ at rates $R_1 \cdots R_n$.

Definition. An *operation* on a file is a seek/rotation followed by a read of some number M_i blocks of the file. An *operation sequence* is a sequence of n operations, one per session. An upper bound $|\phi|$ on the duration of an operation sequence ϕ can be computed based on the session and system parameters. A *workahead-augmenting sequence* is an operation sequence ϕ satisfying

$$M_i B_i > R_i |\phi|$$

for all i, where B_i is the block size of the file read by session S_i (see Figure 14). A set of sessions can be accommodated by CMFS if there is a workahead-augmenting sequence ϕ such that

$$N_i B_i \leq C_i \tag{12}$$

where C_i is the size of the circular buffer used by S_i. The existence of such a sequence is decidable since only finitely many sequence satisfy the above criterion.

For a given workahead-augmenting sequence ϕ and a session S, let $D(S, \phi)$ denote the time after the start of ϕ when S's read is completed. Let T_i denote $w_i(t)/R_i$ (T is the workahead of S_i, expressed in units of time instead of messages). The file system *state* at a given time t is the vector $T = \langle T_1 \cdots T_n \rangle$.

Definition. Let Φ be a set of workahead-augmenting sequences. A state T is *safe* (relative to Φ) iff there exists a workahead-augmenting sequence $\phi \in \Phi$ such that $T_i \geq D(S_i, \phi)$ for all i.

Intuitively, a safe state is one in which the file system can be assured of remaining workahead-positive for all sessions, since it can simply repeat the sequence ϕ over and over.

We now describe the scheduling algorithm used in CMFS. As with CPU scheduling, there are *interactive* and *background* disk requests as well as realtime operations. Let T denote the current state of the system. Let L_i denote the ''workahead limit'' for S_i, measured in time units (this is determined by the buffer space allocated to S_i). These limits must be high enough so that the vector $\langle L_i - X \rangle$ is safe, where X is the duration

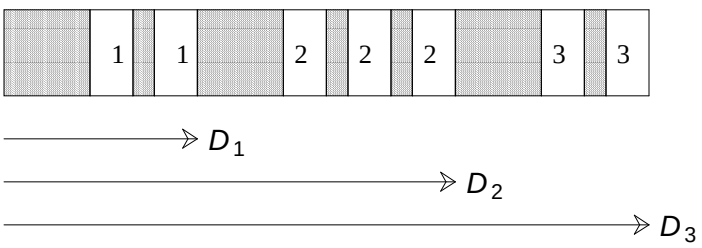

Figure 14: A *workahead-augmenting sequence* is a sequence of disk reads that increases the workahead of all sessions.

(including seek) of the longest possible non-realtime operation; otherwise such an operation could never be handled. Let Φ be a nonempty set of workahead-augmenting sequences, and assume the initial state is safe (relative to Φ). The following procedure is called every time a disk operation finishes:

(1) If $T_i \geq L_i$ for all i: start an interactive request if there is one; otherwise start a background request if there is one.

(2) If there is an interactive request of duration X, and the state with components $T_i - X$ is safe, start the interactive request.

(3) If the system just finished a read for session S_i and $T_i < L_i$ (*i.e.*, the session is not too far ahead), then read the next block of S_i's file. (This policy reduces seeks; it could be modified to switch between files periodically.)

(4) Pick a workahead-augmenting sequence ϕ and perform it. During the sequence, a read for session i is skipped if $T_i > L_i$.

Whenever a read completes, if all sessions satisfy $T_i > L_i$ and there are not non-realtime requests, a timer is started for the earliest time when some session S_i will satisfy $T_i \leq L_i$. If the disk is idle when this timer expires, an operation for the corresponding session is started.

The above algorithm describes the steady-state mode of CMFS. The `reserve()` algorithm for a new session S_n generates one or more workahead-augmenting sequences satisfying Eq. 12, rejecting S_n if there are none. The system then goes into a *startup mode* in which it maintains a state that is safe relative to $S_1 \cdots S_{n-1}$, and schedules reads for S_n when possible (in preference to non-realtime requests). When the system state is safe relative to $S_1 \cdots S_n$, the new client is notified, and the system resumes steady-state mode.

The above design could be generalized to handle variable data-rate files by computing workahead based on timestamps in the file data.

5. RELATED WORK

The notion of a ''linear bounded arrival process'' and its associated workahead function $w(t)$ were proposed by Rene Cruz [10] in the context of multiprocessor interconnection networks. Cruz defines a variety of modules for (de)multiplexing, processing, regulating and generating LBAPs, and develops a calculus for computing delay in networks of these modules. Because of the different domain, Cruz's assumptions are different than ours: devices work at a constant rate, and with fixed scheduling algorithms. Also, there is no notion of logical delay; the goal is to evaluate and control actual delay.

Much work has been devoted to the design of system components (networks, processing, storage) with real-time semantics. For the most part, the resulting systems are compatible with the CM-resource model.

- **CPU scheduling:** An optimal guarantee algorithm for rate-monotonic scheduling on a uniprocessor is known [22]. Various heuristic algorithms for more complex situations (multiple processors, resource contention, *etc.*) have been proposed [20, 44]. An integrated compiler/scheduler (in which the compiler computes code segment execution times) is described by Stoyenko [37].
- **Network communication.** Real-time LAN media access protocols are surveyed in [19]. The goal of end-to-end internetwork connections with meaningful ''quality of service'' parameters is discussed in [29]. The idea of ''logical arrival time'' was

independently developed by Zhang [43], who describes it in terms of ‘‘logical clocks’’. These clocks are used to provide fair queueing of data streams in network gateways.

- **File systems.** Abstractions for multi-media files, including realtime files, are explored in [30, 36]. Issues of disk layout and scheduling are discussed in [1, 42].

Economic approaches have been used for such resource allocation problems as load-balancing [25], network access [17] and file placement [18]. More recently, research has been directed towards ‘‘microeconomic’’ approaches in which system agents are selfish and competitive, and prices vary on demand. This approach has also been used for problems such as routing and load-balancing [12, 26, 41]. It is often possible to show that such systems converge to a globally ‘‘optimal’’ resource assignment.

Finally, the idea of meta-scheduling is present in the design and analysis of some distributed real-time systems [27, 44]. 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.

6. CONCLUSION

The CM-resource model provides a basis for a *meta-scheduler* for a distributed system supporting integrated digital continuous media. The model includes a workload model (linear bounded arrival processes), a model of reservable delay-producing devices (resources), and a protocol that allows a group of resources to negotiate the parameters of end-to-end configurations, in both linear and nonlinear topologies. The resource model allows clients to ensure starvation-free output, and to obtain bounds on end-to-end delay. The amount of buffer space needed on a given host can be computed, eliminating packet loss due to buffer overrun. The combination of these factors greatly simplifies the task of writing CM applications.

Let us return to the goals set forth in Section 1. The CM-resource model fulfills these goals as follows:

Abstraction: The model defines an abstract interface that can be implemented for a range of system components (CPU, network, file system). Many existing scheduling policies (FIFO, FDDI timed-token protocol, *etc.*) can be encapsulated by this interface.

Coexistence: By allowing system components to work ahead, response for bursty non-realtime clients can be improved. Furthermore, any unused portion of a reservation can be used by non-realtime traffic.

End-to-end guarantees: The model allows true end-to-end performance guarantees. The compound session establishment protocol allows negotiation of parameters (workahead, delay bounds) among all the components of an end-to-end session.

The CM-resource model is therefore a viable basis for the development of distributed computer systems offering integrated digital continuous media. Considerable work remains to be done, however, in investigating the use of the CM-resource model in the design of operating systems, file systems, network design and protocols, and in application programming.

ACKNOWLEDGEMENTS

The CM-resource model is a group effort. Martin Andrews and Robert Wahbe developed the basic model (LBAPs, resources, and linear compound sessions). Ralf Herrtwich found the correct definition of logical delay and proved its additive property.

Ramesh Govindan found the conditions for non-starvation. Eric Barr formulated the deadline-workahead scheduling policy. Steve Lucco suggested the algorithm for tree-structured compound sessions. Ralf and Eric also gave careful readings of a draft of the paper.

REFERENCES

1. C. Abbott, ‘‘Efficient Editing of Digital Sound on Disk’’, *J. Audio Eng. Soc. 32*, 6 (June 1984), 394.
2. D. P. Anderson, R. G. Herrtwich and C. Schaefer, ‘‘SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet’’, Technical Report 90-006, International Computer Science Institute, Feb. 1990.
3. D. P. Anderson, R. Govindan and G. Homsy, ‘‘Abstractions for Continuous Media in a Network Window System’’, UC Berkeley, EECS Dept., Technical Report No. UCB/CSD 90/596, Sep. 1990.
4. D. P. Anderson, S. Tzou, R. Wahbe, R. Govindan and M. Andrews, ‘‘Support for Continuous Media in the DASH System’’, *Proc. of the 10th International Conference on Distributed Computing Systems*, Paris, May 1990.
5. D. P. Anderson, L. Delgrossi and R. G. Herrtwich, ‘‘Process Structure and Scheduling in Real-Time Protocol Implementations’’, *Proc. Kommunikation in Verteilten Systemen*, Feb. 1991. also Technical Report 90-021, International Computer Science Institute, June 1990.
6. D. P. Anderson, R. Govindan and G. Homsy, ‘‘Abstractions for Continuous Media in a Network Window System’’, *International Conference on Multimedia Information Systems*, Singapore, Jan. 1991.
7. M. Andrews, ‘‘Guaranteed Performance for Continuous Media in a General Purpose Distributed System’’, Masters Thesis, UC Berkeley, Oct. 1989.
8. B. Arons, C. Binding, K. Lantz and C. Schmandt, ‘‘The VOX Audio Server’’, *Multimedia ’89: 2nd IEEE COMSOC International Multimedia Communications Workshop*, Ottawa, Ontario, April 20-23, 1989.
9. P. T. Brady, ‘‘A Statistical Analysis of On-Off Patterns in Sixteen Conversations’’, *Bell System Technical Journal 47*, 1 (Jan. 1968).
10. R. L. Cruz, ‘‘A Calculus for Network Delay and a Note on Topologies of Interconnection Networks’’, Ph.D. Dissertation, Report no. UILU-ENG-87-2246, University of Illinois, July 1987.
11. S. E. Deering and D. R. Cheriton, ‘‘Multicast Routing in Datagram Internetworks and Extended LANs’’, *Trans. Computer Systems 8*, 2 , 85-110.
12. D. Ferguson, Y. Yemini and C. Nikolaou, ‘‘Microeconomic Algorithms for Load Balancing in Distributed Computer Systems’’, *Proc. of the 8th International Conference on Distributed Computing Systems*, San Jose, California, June 1988.
13. K. A. Frenkel, ‘‘The Next Generation of Interactive Technologies’’, *Comm. of the ACM 32*, 7 (July 1989), 872-881.
14. R. G. Herrtwich, ‘‘Time Capsules: An Abstraction for Access to Continuous-Media Data’’, *11th Real-Time Systems Symposium*, Orlando, Dec. 1990.
15. R. Jain, K. K. Ramakrishnan and D. M. Chiu, *Congestion Avoidance in Computer Networks with a Connectionless Network Layer%R DEC-Tech. Rep.-506*, DEC Corp., Aug. 17, 1987.
16. R. Karp, ‘‘Reducibility Among Combinatorial Problems’’, in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (editor), Plenum Press, New York, 1972, 85-103.

17. J. F. Kurose, M. Schwartz and Y. Yemini, ‘‘A Microeconomic Approach to Optimization of Channel Access Policies in Multiaccess Networks’’, *Proc. of the 5th International Conference on Distributed Computing Systems*, .

18. J. F. Kurose and R. Simha, ‘‘A Microeconomic Approach to Optimal File Allocation’’, *Proc. of the 6th International Conference on Distributed Computing Systems*, Cambridge, Massachusetts, .

19. J. F. Kurose, M. Schwartz and Y. Yemini, ‘‘Multiple-Access Protocols and Time-Constrained Communication’’, *ACM Computing Surveys 16*, 1 , 43-70.

20. D. W. Leinbaugh and M. Yamini, ‘‘Guaranteed Response Time in a Distributed Hard Real-Time Environment’’, *IEEE Trans. on Software Eng. 12*, 12 (Dec. 1986), 1139-1144.

21. T. D. C. Little and A. Ghafoor, ‘‘Synchronization and Storage Models for Multimedia Objects’’, *IEEE Journal on Selected Areas in Communications 8*, 3 (Apr. 1990), 413-427.

22. C. L. Liu and J. W. Layland, ‘‘Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment’’, *J. ACM 20*, 1 (1973), 47-61.

23. G. Loy, ‘‘Designing an Operating Environment for a Realtime Performance Processing System’’, *Proceedings of the 1985 International Computer Music Conference*, Burnaby, B.C., Canada, 1985, 9-13.

24. A. C. Luther, *Digital Video in the PC Environment*, McGraw-Hill, 1989.

25. P. Ma, E. Lee and M. Tsuchiya, ‘‘A Task Allocation Model for Distributed Computing Systems’’, *IEEE Trans. on Computers 31*, 1 (Jan. 1982), 41-47.

26. T. W. Malone, R. E. Fikes, K. R. Grant and M. T. Howard, *The Ecology of Computation*, North Holland, 1988.

27. C. W. Mercer, Y. Ishikawa and H. Tokuda, ‘‘Distributed Hartstone: a Distributed Real-Time Benchmark Suite’’, *Proc. of the 10th International Conference on Distributed Computing Systems*, Paris, May 1990.

28. N. Ohta, M. Nomura and T. Fujii, ‘‘Variable Rate Video Encoding Using Motion-Compensated DCT for Asynchronous Transfer Mode Networks’’, *IEEE International Conference on Communications*, 1988.

29. G. M. Parulkar and J. S. Turner, ‘‘Towards a Framework for High-Speed Communication in a Heterogeneous Networking Environment’’, *IEEE Network 4*, 2 (March 1990), 19-27.

30. J. M. Roth, G. S. Kendall and S. L. Decker, ‘‘A Network Sound System for UNIX’’, *Proceedings of the 1985 International Computer Music Conference*, Burnaby, B.C., Canada, Aug. 19-22, 1985, 61-67.

31. C. Schmandt and B. Arons, ‘‘A Conversational Telephone Messaging System’’, *IEEE Transactions on Consumer Electronics CE-30* (August 1984).

32. C. Schmandt, ‘‘Employing Voice Back Channels to Facilitate Audio Document Retrieval’’, *ACM SIGOIS & IEEECS TC-OA*, Palo Alto, CA, March 1988.

33. C. Schmandt and M. A. McKenna, ‘‘An Audio and Telephone Server for Multi-Media Workstations’’, *Proc. 2nd IEEE Conf on Computer Workstations*, Santa Clara, CA, 1988.

34. K. C. Sevcik and M. J. Johnson, ‘‘Cycle Time Properties of the FDDI Token Ring Protocol’’, *IEEE Trans. on Software Eng. 13*, 3 , 376-385.

35. L. Sha, R. Rajkumar and J. P. Lehoczky, ‘‘Priority Inheritance Protocols: An Approach to Real-Time Synchronization’’, Tech. Report CMU, Carnegie-Mellon University, Pittsburgh, PA-CS-87-87-181, Nov. 1987.

36. D. Steinberg and T. Learmont, ‘‘The Multimedia File System’’, *Proc. 1989 International Computer Music Conference*, Columbus, Ohio, Nov. 2-3, 1989, 307-311.

37. A. D. Stoyenko, ‘‘A Schedulability Analyzer for Real-Time Euclid’’, *Proc. 1987 IEEE Real-Time Systems Symposium*, Dec. 1987.

38. D. B. Terry and D. C. Swinehart, ‘‘Managing Stored Voice in the Etherphone System’’, *Trans. Computer Systems 6*, 1 (Feb. 1988), 3-27.

39. J. N. Ulm, ‘‘A Timed Token Ring Local Area Network and its Performance’’, *Proc. 7th Conf. on Local Computer Networks*, Feb. 1982, 50-56.

40. A. Valenzano, P. Montuschi and L. Ciminiera, ‘‘Some Properties of Timed Token Medium Access Protocols’’, *IEEE Trans. on Software Eng. 16*, 8 , 858-869.

41. C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephard and S. Stornetta, ‘‘Spawn: A Distributed Computational Economy’’, *Technical Report, Xerox PARC*, May 8, 1989.

42. C. Yu, W. Sun, D. Bitton, R. Bruno and J. Tullis, ‘‘Efficient Placement of Audio Data on Optical Disks for Real-Time Applications’’, *Comm. of the ACM 32*, 7 (1989), 862-871.

43. L. Zhang, ‘‘A New Architecture for Packet Switching Network Protocols’’, Ph.D. Thesis, MIT, July 1989.

44. W. Zhao, K. Ramamritham and J. A. Stankovic, ‘‘Preemptive Scheduling Under Time and Resource Constraints’’, *IEEE Transactions on Computers 36*, 8 (Aug. 1987), 949-960.

45. ‘‘Draft Proposed Standard, FDDI Token Ring Station Management (SMT)’’, ASC X3T9.5, Rev. 6.2, May 1990.

46. ‘‘American National Standard, FDDI Token Ring Media Access Control (MAC)’’, American National Standards Institute X3.139-1987, 1987.