システムソフトウェアと
オペレーティング・システム 63－8
（１９９４．３．２２）

Real-Time Mach 3.0 上の連続メディア処理のための協調サーバ群の設計と実験†

西尾 信彦‡

慶應義塾大学環境情報研究所

あらまし　連続メディアデータを分散環境下で実時間性を保証しつつ処理するため、メディアストリーム同期機構、予約可能な共有メモリ管理、動的なQOS(サービスの質)制御を備えた各種サーバ群の協調的なふるまいに関するモデルの設計を行ない、そのプロトタイプを実装し実験を行なった。連続メディアのストリームをメディアサーバと同期する同期ストリームと非同期な非同期ストリームの両者を結合したハイブリッドストリームを用いた実装の実験とそれにより得られた知見および今後の展望について報告する。

キーワード　リアルタイム, マイクロカーネル, QOS 制御, マルチメディア, 連続メディア, 分散システム

Design and Experiments of Cooperative Servers for Continuous Media Processing on Real-Time Mach 3.0

NISHIO, Nobuhiko

vino@sfc.keio.ac.jp, Keio Institute of Environmental Information, Keio University,
5322, Endo, Fujisawa-shi, Kanagawa, 252 Japan,

Abstract　We have been developing real-time servers which process continuous media such as video and voice in distributed environment. In this article, we design cooperative activity model for continuous media processing servers, which are equipped with media stream synchronization, shared buffer management and dynamic QOS - quality of service control. We implement and have their performance evaluation of some prototype servers, which involve synchronous stream, asynchronous stream and hybrid stream. Future research directions are presented along with the experiment results.

Keyword　Real-Time Processing, Micro-kernel, QOS Control, Multimedia, Continuous Media, Distributed System

†この研究は, 情報処理振興事業協会 (IPA) が実施している開放型基盤ソフトウェア研究開発評価事業「マルチメディア統合環境基盤ソフトウェア」プロジェクトのもとに行なわれました. ‡ 開放型基盤ソフトウェア湘南藤沢キャンパス研究室の研究員として IPA に登録されている.

1 はじめに

われわれは動画や音声などの時間にしたがって連続的に変化していくような、連続メディア (Continuous Media) を扱うために、実時間性を考慮した基盤ソフトウェアを開発/評価する研究を行なっている。本稿では実時間カーネル上で連続メディアアプリケーションとの間で動作する連続メディア処理のための各種サーバ群をいかに実時間性を守って協調/同期して動作させるかの研究について述べる。

連続メディア環境において実時間性を実現するためには、OS など基本ソフトウェアのレベルから、実時間性を保証できなければならない。通常の UNIX システムでは、連続メディアの再生を行うと、動画や音声の再生が途切れたり各メディア間の同期がずれ、実時間性を実現することは困難である。ここでわれわれは、実時間カーネルとして Real-Time Mach 3.0[1] を採用することにより、UNIX ドメインとは独立して Mach カーネルの上で自分でスケジューリングポリシーを選択し、周期/非周期の実時間スレッドをデッドラインハンドラつきで生成できるようにした。実際のハードウェアとしては IBM/PC AT 互換機である、Gateway2000 4DX2-66V(i486DX2/66MHz, 16MBRAM, IDE 340MBHD × 2) を用いた。RT-Mach は MK83 にリアルタイム機能拡張したものである。

また連続メディア処理のためのモデルとして、各種の連続メディアを個別に扱うサーバ群を Mach カーネルの上で協調動作させることにした [2]。このモデルではサーバ群の中にメディアサーバと呼ばれるものがあり、どのアプリケーションもメディアサーバを通してのみ連続メディア処理のためのセッションを確立する。

その他の各種サーバの中には、連続メディアデータのためのファイルサーバやウィンドウサーバ、システムの分散環境対応のための実時間ネットワークプロトコルサーバ [3] を始めとして、サウンドボードとの入出力をになうサウンドサーバやアナログのビデオ信号のデジタイズをするビデオキャプチャーボードを制御するビデオサーバ、ストリームを流れているデータを JPEG や MPEG などで圧縮/伸長を行なったり、他のストリームデータと比較やミキシング、エッジ検出や音声データの周波数解析をするような特殊なフィルターサーバに至るまでさまざまなものを含めて考えている。

中でもわれわれは Mach 3.0 の外部ページャを用いて実現したファイルサーバである連続メディアベース [4] や、X Window System を実時間のマルチスレッドで書き直したり [5]、それにメモリ共有による Mach IPC のリクエストポートをつける研究も同時に進行させている。本稿はそれらの各種サーバを協調動作させるためのフレームワークの実験に位置づけられる。

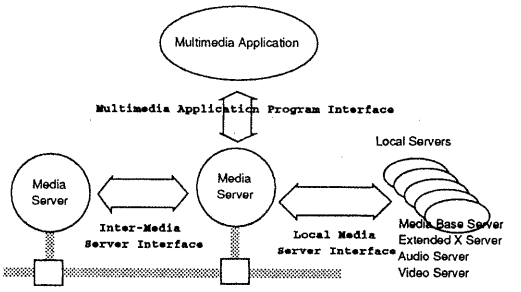

図 1: メディアサーバ I/F 構成

以下では、まずわれわれの採用している基本モデルについて簡単に説明し、それを素直に実装したプロトタイプの例を経て、その改良モデルについて説明し、その実験結果について述べていく。

2 メディアサーバ基本モデル

メディアサーバを中心にすえたモデルを設計するためにわれわれが考慮したのは、

1. 連続メディアの Inter-/Intra-ストリームが同期する実時間性があること、
2. 各種のアプリケーションの要求に応じてさまざまな形態のセッションを自由に組み立てられること、
3. セッション確立前の資源見積りができること (admission 制御)、
4. セッション確立後の QOS 制御 (動的な資源制御) ができること、
5. 効率がよく、資源の usability が高いこと、

である。Inter-stream 同期は音声と動画のタイミングをずれないように保証することに代表される問題である。Intra-stream 同期は動画の各フレームが一定周期で再生されたり、音声が途中で切れたりしないようにすることである。さまざまな形態のセッションを組み立てるとは、CCD カメラからのデータを再生するのと、途中で JPEG 圧縮しながらハードディスクに格納するのを別々のアプリケーションで実装するのではなく、もっと細かな部品から組み立てられるようにすることを意味する。セッション確立前の資源見積もりは、要求されたセッションに必要なプロセッササイクル、物理メモリ、ネットワークバンド幅などを事前に見積って、現在それが賄えるかどうかを判定する機能である。これを admission 制御と呼ぶ。

動的な QOS 制御は特に重要な要件である [8]。実際に事前に確保した資源が多過ぎたり少な過ぎたりした場合にそれを調整したり、システムに一過性の負荷が加わった場合に対処するための機能であり、われわれが対象とするいわゆるソフトリアルタイム処理では不可欠な機能である。また、予約した QOS からできる限りはずれないことや、そのジッタ特性が軽微なことなどが望ましく、できる限り無駄に資源を使用することがないよう、とかくハードリアルタイムシステムにありがちな悲観的な見積もりによる usability の低下を招かないように留意した。

そこでわれわれは以下の方式を採用した。

- メディアサーバを中心とする複数のサーバから構成する。
- メディアサーバはホストにひとつづつ存在し、そのホストの資源を管理する。
- アプリケーションとの I/F はメディアサーバがになう。
- 各種サーバはメディアサーバとの I/F をもち、それぞれの内部の実装は自由である。
- メディアサーバは要求されたセッションに必要な資源の見積もりを立て、admission 制御をする。
- メディアサーバは各種サーバを連結して生成したストリームの実時間性の保証に責任がある。
- メディアサーバは単一セッション内のストリーム間の同期に責任がある。
- メディアサーバはアプリケーションから要求される QOS を保証するために、ストリームの動的な性能評価をして各種サーバとの I/F により制御する。
- アプリケーションライターには各種サーバをメディアオブジェクトという「部品」で表象してそれを連結する抽象化を提供する。
- 分散環境への拡張のためにメディアサーバ間通信のプロトコルを用意する。

メディアサーバは自分への要求により、その他の各種サーバと通信をしてそれらからなるデータストリームを確立して、連続メディアデータの実時間性を保証しつつそのサービスの品質 (Quality of Service, QOS) の制御 [6] を行なう。データストリームを確立するときにそれに必要な資源を見積ってその確保をして admission 制御を行ない、ストリームが確立してからは、そのセッションの実時間性を保つ責任があり、そのための動的な性能評価や QOS 制御をする。

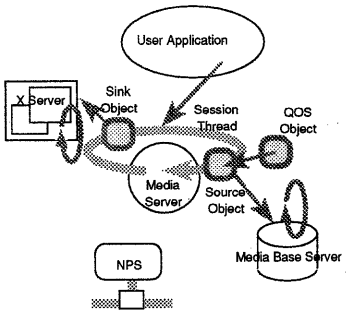

図 2: メディアサーバ基本モデル

3 基本モデルに基づく単純な実装フレームワーク

そこで、前節で提示したモデルに従って実装したフレームワークについて説明する。

3.1 メディアサーバ内のメディアオブジェクト

メディアサーバのアプリケーション I/F は基本的にメディアサーバ内にあるメディアオブジェクトを操作することになる。メディアサーバは新たなセッションを生成する要求がくると、その要求に応じてこの各種サーバの管理する実体を代表するクラスのオブジェクトを連結してストリームを生成し、その Intra-stream 同期に応じた周期スレッドを生成する。たとえば、ハードディスク上のムービーデータを再生するセッションの場合、音声と動画の二つのストリームが形成され、それぞれの周期が異なった二つの周期スレッドが生成される。周期スレッドはセッション内のデータストリームを表わすことになる。

これらのオブジェクトはメディアサーバのアドレス空間内に存在するわけだが、それぞれのオブジェクトは自分の所属するクラスに応じてバッキングサーバが存在する．たとえばファイルサーバ内の連続メディアデータを表わすメディアオブジェクトの場合は、バッキングサーバはファイルサーバであり、動画を表示するウィンドウの場合は拡張されたウィンドウサーバである。各種バッキングサーバに対応してメディアオブジェクトのクラスを追加されていくことになる。

メディアサーバ内の一つのメディアオブジェクトに対応してバッキングサーバ内にも一つのオブジェクトないしはセッション、スレッドなどの実体が生成される．各種バッキングサーバによりどのような実体が生成されても構わない．これら二つ（セッションサーバ内のメディアオブジェクトとバッキングサーバ内の実体）の間で RPC を行ない全体として一つのオブジェクトのようにアプリ

ケーションに抽象化して見せる。

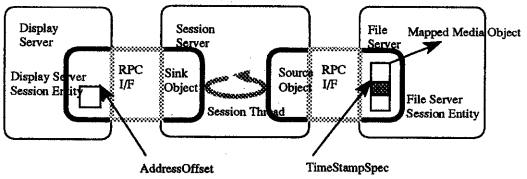

図 3: 同期ストリームの構成

3.2 単一メソッド起動によるストリームの実現

多少簡略化されているが以下にメディアサーバ内のメディアオブジェクトが generic に持つべきメソッドを挙げている。

- ReserveQOS(QOSSpec) そのオブジェクトに QOSSpec で指定された QOS を実現するための資源を確保させる。
- ChangeQOS(NewQOSSpec) そのオブジェクトに現在指定されている QOS を NewQOSSpec に変更し，そのための資源の追加確保もしくは解放を行なわせる。
- MapNextMDU(TimeStampSpec, DestObj, &MappedAddress) 最上流オブジェクトでは、そのオブジェクトにある TimeStampSpec で指定された MDU(Media Data Unit)[1]をストリームが次に流れ込むメディアオブジェクト DestObj のバッキングサーバのアドレス空間の空いているところへマップし、そのマップしたオフセットを MappedAddress に返す。それ以外のオブジェクトでは MappedAddress が入力引数としても働く。

これらのメソッドをたとえばファイルサーバからデータを読んでウィンドウに描画する場合、メディアサーバ内で生成された周期スレッドが以下のように起動していく。

```
/* ストリーム確立のための資源予約、失敗したら
admission 制御ではねる */
FileMediaObject->ReserveQOS(QOSSpec);
DisplayMediaObject->ReserveQOS(QOSSpec2);
TimeStampSpec->Init(QOSSpec);

/* 以下は周期スレッドが起動される毎の処理 */
if(TimeStampSpec->IsOutOfRange(FileMediaObject)){
  FileMediaObject->
    MapNextMDU(TimeStampSpec,
               DisplayMediaObject, &Address);
  DisplayMediaObject->
    MapNextMDU(TimeStampSpec, NULL, &Address);
  TimeStampSpec->Advance(QOSSpec);
}
```

[1]動画データではいわゆるフレームに、音声データでは一回の操作で行き来するデータの単位量に相当する。

この周期スレッドが自分の周期内で処理を完了できない場合はデッドラインハンドラが自動的に起動され、どのメソッド実行中かにより適切なメディアオブジェクトを選択して ChangeQOS(AppropriateQOSSpec) メソッドを起動する.

このフレームワークは、ストリーム内のすべてのデータの移動を統一的に単純な一種類のメソッド起動により行なうことにより単純化して、全体としては見とおしがよい。また重要なのは、再生するすべての動画フレームに対して毎回メディアサーバがそのタイムスタンプを指定する方式であり、以後ストリームとメディアサーバが同期して動作することから同期ストリームと呼ぶことにする。

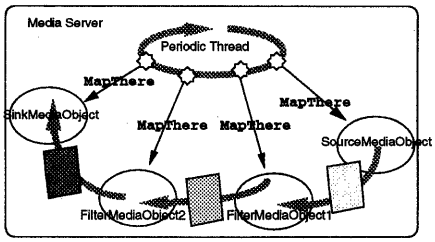

図 4: 単一メソッド起動

4 並列/先行制御を追求した実装フレームワーク

前節の方式によるフレームデータ描画は周期スレッドと各種サーバでの処理をすべて同期させるものであるために、並列性が一切認められない。たとえば、ディスクからのフレームデータの読み込みとウィンドウへの描画処理とは同時に処理が可能である。またこのような並列性の追求に加えて、このファイルサーバのような挙動の予測不可能な要素が大きいバッキングサーバがストリームの中に入ってきたとき先行制御によりバッファリングしておくことにより処理のジッタ傾向を吸収することが可能である。ここでは以前の方式を踏まえて更に並列/先行制御を追求した実装フレームワークを追求する。

4.1 同期ストリームと並列/先行制御

同期ストリームでの周期スレッドのはたらきは、データを出力するテンポを教える heart beat のようなもので、各バッキングサーバにそのとき処理しているフレームを可能な限りすみやかに出すように教えるものである。

しかし、たとえば各ステップでの遅延が絶対的に大きいかもしくは、挙動が予測不可能である場合にはスムーズな再生には先読みが有効で必要になるが、どうしてもその先行制御のタイミングはメディアサーバ内の周期スレッドとテンポとはずれたものになってしまう。そしてその挙動に予測不可能性がある場合には、正確なテンポをきざむ周期スレッドと先行制御とのあいだには、単に固定的な位相のずれを入れるだけでは対応できない。そのため、この方式にそのまま先行制御を導入すると、メディアサーバが関知しない/できないところでそれを行わざるをえなくなってしまい、それは望ましいものではない。

その解決法としてメディアサーバ内のスレッドの処理のテンポに間に合う (予測可能 and/or 遅延の小さい) 部分と、(予測不可能 and/or 遅延の大きい) 先行制御が必要な部分とに分けてそれぞれに異なった処理を施し、両者をハイブリッドに結合するという案である。前者の部分はメディアサーバ内の周期スレッドのきざむテンポに合わせてフレームマップを行なう同期ストリームを、後者は先行制御を伴ない、最初のレート指定以降はメディアサーバとは非同期に自律的な一定レートのストリームを確立して実現する。後者を今後、メディアサーバと非同期に動作することから非同期ストリームと呼び、両者を結合したストリームをハイブリッドストリームと呼ぶ。

一つのストリームを上記の2パートに分けて管理する。具体的にはディスクからストアドメディアを読み込み、それを codec にかけて、出力デバイスにくべて再生する場合、最後の出力デバイスに出すタイミングのみを周期スレッドがとり、それより上流をすべて非同期ストリームとして実現し両者を結合するわけである。

最下流部の方式については、以前に説明しているので、上流部の非同期ストリームについて説明する。

4.2 非同期ストリーム

この非同期ストリームは、毎回タイムスタンプを指定してひとつひとつフレームを要求する同期ストリームとは異なり、最初にレート (を含むいわゆる QOS 値) を指定した後は、自律的にバッファにデータを供給することで特徴づけられる。

具体的な実装は上流 (producer) と下流 (consumer) の間に適切な量のリングバッファを提供し、両者が同期するように動作させる。このリングバッファにアクセスする処理は各種バッキングサーバ内の非周期リアルタイムスレッドで、メディアサーバ内の周期スレッドのバックグラウンドでかつ非リアルタイムスレッドより優先させて行なう。スケジューリングポリシーは FixedPriority 方式を採用する。

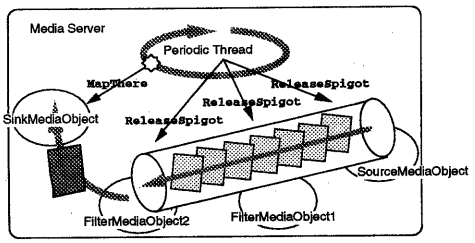

図 5: ハイブリッド (非同期/同期) ストリーム

4.3 非同期ストリームの共有バッファ管理

各種バッキングサーバは期待された QOS を満たすのに必要な量[2]のバッファをメディアサーバに用意してもらい、それをアクセスするすべてのサーバのアドレス空間に共有させる。つまり、ストリーム毎にメディアサーバ内にそのストリームに関連する全バッキングサーバがアクセスできる黒板が用意される。その黒板の中から各サーバが (ライブラリ経由で) 自分の使用可能領域を割当ててロックし、自分の処理を行なう。自分の処理が完了すると、どの処理が終ったかをロックに書き込み解放する。すると、今度は自分の下流側のサーバがそれを用いて処理を始める[3]。

たとえば、ファイルサーバ A から読み込んだフレームを JPEG サーバ B が解凍して、X サーバに出力するとすると、あるバッファ領域のロック状態は、

フリー → サーバ A 処理中 → サーバ A 処理完了 → サーバ B 処理中 → サーバ B 処理完了 → フリー

JPEG サーバ B は解凍によりサイズが変化するので、別の共有バッファ領域が、

フリー → サーバ B 処理中 → サーバ B 処理完了 → X サーバ処理中 → X サーバ処理完了[4] → フリーのように変遷する。

このようにして、常に一定の量しかメモリ資源を使わないように限定でき、かつメモリのマップの手間も節約できるという利点がある。今後この共有バッファを Cyclic Shared Buffer(CSB) と呼ぶ。

[2] バッファの量の算出は各種バッキングサーバに任せられる。

[3] このような共有メモリではややメモリ保護機能が弱まるが性能を重視した。

[4] 最後の X サーバが解放するタイミングは (現時点での実装では) 判定が困難で、二巡目のフレームを X サーバにマップしたときに隠れたフレームを解放している。

すべてのCSBをメディアサーバがまず獲得して、それから使用するバッキングサーバにマップすることにしたがその理由は、各バッキングサーバが処理状況に応じてバッファにつけるマークをメディアサーバが参照してストリームの全体性能を評価できるようにするためである。

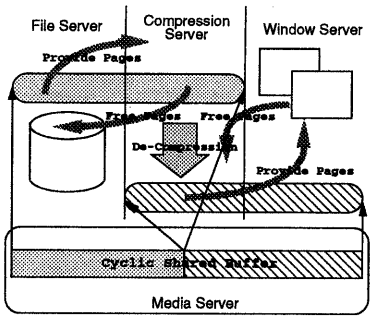

図 6: 共有バッファ機構

4.4 ハイブリッドストリームの動的なQOS制御

非同期ストリームではいずれかのバッキングサーバが処理すべきデータが特定量より滞留し始めた場合に、メディアサーバへその旨を通信し、メディアサーバは最上流へ適切な調整指示を出す。その調整指示は上流から順に下流まで伝播させ全体での調整をとる。また、メディアサーバ内の周期スレッドも同期ストリームについてのいくつかの実行時性能評価を行ないそれをトリガーとしたQOS制御も同時に進行する場合もある。周期スレッドの実行時性能評価としては以下のようなものが考えられている。

- デッドラインハンドラを契機とする、
- 自らの周期を実測して判定する、
- CSBの各単位フレームに各バッキングサーバがつけたマークで全体状況を把握して

行なうといったものである。

4.5 ハイブリッドストリームによるQuickTime Playerの実装

このハイブリッドストリームを用いてQuickTime Playerを実装した。これは、Apple社の連続メディアファイル形式であるQuickTime[7]の特定の形式(8bit Color, 8bit/22KHz Monaural Sound, Raw/JPEG Codec)のファイルをUNIXファイルシステム上に格納しておいて、X Windowやサウンドサーバで再生するものである。

われわれは、RT-Mach 3.0上でQuickTime Playerをメディアサーバ(qtplay)に見立てて、MIGを用いてUXサーバ上に作成した簡易なファイルサーバ(qt_server)、MediaVision社のProAudio16サウンドカードを駆動できるサウンドサーバ(pas_serv)、拡張されたX Windowサーバ(machX)といった各種サーバと上記の方式で協調動作するプロトタイプを実装した。アプリケーションとメディアサーバが未分化な実装であり、メディアサーバのAPIはまだ存在しない。

Qtplayは一つのQuickTimeファイルの再生に関して、音声と動画の二つのストリームを二つの周期の異なった(音声1sec、動画1/秒間フレーム数sec)周期スレッドではる。音声スレッドの方を優先度を上げて、FixedPriorityポリシーでスケジューリングした。

簡易ファイルサーバであるqt_serverはMIGによるMachIPCで起動できるシングルスレッドサーバで、QuickTimeファイルについてそのタイムスタンプを指定してそのフレームデータを読み込み、それを任意のアドレス空間にマップすることができる。

サウンドサーバはリアルタイムMIGを用いてMach Real-Time IPCでクライアントスレッドの優先度を伝播させて起動できるサーバである。

拡張されたX Windowサーバは、通常のソケットI/F以外に、MachIPCによるクロスアドレス空間でのページマップによるXプロトコルを実装している。このおかげで特にイメージデータの描画などで、クライアントからサーバへのイメージデータのコピーが必要なくなり動画データ向きの高速なイメージ描画が可能である。

最初にqtplayからファイルサーバにファイル名とフレームレートを添えてストリームのオープン要求を出す。ファイルサーバはストリームを代表するポートと先行制御用に必要なバッファの大きさを返す。

Qtplayはそのファイルについての情報を取得して、ファイルサーバが次にデータをわたす相手を選定する。それがまだ非同期ストリーム内なら、そこへファイルサーバからのストリームをコネクトするように要求する。そこでもそのサーバでのストリームを代表するポートを生成してそこで必要なバッファの大きさを見積り、qtplayへはストリームへのポートとバッファの大きさを返す。非同期ストリームがファイルサーバだけで終りならqtplayがデータをマップされるように準備する。

非同期ストリームの最後までコネクトできると、qtplayは各自の必要量のバッファを自分のアドレス空間に獲得してそれを各バッキングサーバのアドレス空間にマップしていく。その後、qtplayはファイルサーバのストリー

ムポートに使用可能なバッファのリストをメッセージにして非同期に送信しておく。

以上で非同期ストリームを動かす準備が完了した。続いてqtplayからファイルサーバにデータストリームを開放しCSBをfill-upする同期的な要求をする。このとき、引数としてファイルサーバがストリームをコネクトしている次のサーバへのポートをメディアデータのターゲットポートとして渡す。

ファイルサーバは、そのストリームの先行スレッドを生成して動かすと、そのスレッドがRTスレッドなのでMIGのリプライよりも先に動き始める。先行スレッドは自分のストリームポートから空のバッファを獲得して、指定されたQOSに応じたメディアデータをそこに格納する。その処理の完了したバッファ(のリスト)をメッセージにして(非同期に)ターゲットポートに送信する。

先行スレッドは自分の使用可能なバッファがなくなるなど待ち状態になると、自分のストリームポートに対して受信待ちに入る。するとファイルサーバのMIGのリプライ処理が動き出し、制御がqtplayに返る。

Qtplayは非同期ストリームが次にコネクトしているサーバがあれば、そこにやはりデータストリームを開放する(同期的な)要求をする。この場合、二つのポートを引数として渡すことがある。もしそのサーバが使用を終了したバッファを開放するようなことがあるならばその先として適切なストリームポートと、更に自分の処理が完了したバッファを次に使うサーバのターゲットポートである。サーバ内では先行スレッドを生成し自分のストリームポートからメディアデータを順にとって処理していく。自分のストリームポートは自分の上流サーバのターゲットポートである。処理すべきCSBが尽きるとやはり自分のストリームポートに新規CSB待ちになって、qtplayに制御が返る。

このように順にフォワードされていったバッファが(それを待っている上流のサーバに)開放されると、それを使って先行処理が更に進んでいく。

非同期ストリームの最後を開放する呼出しがqtplayに返ってきたときには、存在するすべての先行制御用のバッファがfill-upされており、同期ストリームの周期スレッドを動かせるタイミングになる。非同期ストリームの最後のサーバにとってのターゲットポートはqtplay内の周期スレッドに対応するものとなる。

4.6 ハイブリッドストリームによるQuickTime Playerの実験結果

本実装では、ファイルサーバ以外にJPEGなどのCodecの処理を行なうサーバを非同期ストリームに接続することもできるが、ソフトウェアCodecの性能の面から実験ではそれを除きファイルサーバとqtplayの間が非同期ストリームで、qtplayと拡張X Windowサーバの間が同期ストリームとした。

実験では、120 × 160で8bit256色カラー、10fpsのQuickTimeファイルデータをIDE HDから再生させてみた。このとき前述のCSBの各バッファにつけられたマークの統計を取り先行制御が働いていることを確認した。この場合、同期ストリームで周期スレッドがリズームしたときマップすべきデータがないと、先行制御は効果がないことを示す。われわれは先行処理用に1セッションで4フレーム読み込めるCSBを用意した．この内，メディアサーバ内の周期スレッドがリズームしたときには常時2つが先読みされており，1つがX Windowサーバに使用されており，残る1つがフリーであった．再生性能限界を同期ストリームのみによるQuickTime Playerと比較するため30fpsのデータを用いたところ10秒間のデータを1分ほど連続再生させて、ハイブリッドが20fps、同期が13fps程度であった。これは、主にディスクへのアクセスが他のウィンドウ描画などの処理と並行動作でき、周期スレッドのアイドル時間にも先行処理として行なわれたことによる。ただし、どちらのファイルサーバもファイルのアクセスに関しては同じもので絶対的に遅く処理時間も一定していない。

動的なQOS制御はいくつかの方法を試みた。まず最初に、qtplayの周期スレッドのデッドラインミスをトリガーに性能統計をとり、QOSの変更要求をファイルサーバに出すようにした。これは、Real-Time Mach 3.0のリアルタイムスレッドモデルがデッドラインミス後に何も調整しないでひたすら負債をとり返そうとするcatch-upモードでできているために、実状以上にデッドラインミスの回数が増えてしまい、極端にQOSが下げられてしまい使いものにならなかった。これは、あえてatch-upをしないなどのリアルタイムスレッドモデルの拡張が必要でその研究も並行させている[10]。

次に試してみたのが、周期スレッドが自分の実際の周期を計測し、最近何回かの平均からそのときの適正QOS値を見積って調整する方式である。

この他の動的QOS制御として非同期ストリーム中のサーバでバッファが滞留するものがメディアサーバに申告する方式が考えられるが，この実装ではメディアサーバが唯一の滞留監視サーバにあたり，実はこれは全バッファのマークの統計採集に含まれる処理であるために特には行なわなかった。

これらの動的QOS制御の性能は、システムの限界性能あたりで制御しようとすると(i)QOSの変更要求が頻繁に飛び過ぎることがあったり、(ii)たとえば負荷を下げようとして本来ファイルに格納されているフレームレート

より下げて飛ばし読みを始めると、飛ばさないでアクセスするよりもファイル読み出しの性能が下がってしまい、全体として期待した性能に安定させられなかった。(ii) は、ファイルサーバの性能向上が見込まれ今後の検討課題となった。

5 まとめと今後の研究課題

本稿では、連続メディア処理のためのサーバを各種用意し、それらを実時間性を守って協調/同期させることを目的とするフレームワークの設計をし、そのプロトタイプを QuickTime Player として実装した。この中で、メモリ資源の使用量を確定させて、それを関連サーバに共有マップすることにより効率よく使い回せる Cyclic Shared Buffer を提案し、それの動作をプロトタイプにより確認した。また、連続メディアのストリームをメディアサーバと同期するものと非同期に先行制御するものに分けてそのハイブリッドにより再生効率などが向上できることを確認した。

今後予定している研究課題を以下に挙げる。

- システム全体での資源予約管理機構の研究。この中には、リアルタイム IPC や、ユーザレベルスレッドパッケージ [9] などが含まれる。
- ファイルサーバの絶対的な性能向上を追求する。それは、データ転送バンド幅の拡大と、その predicatable な予約配分の追求が含まれる。
- アプリケーションライターに提供するメディアオブジェクト抽象に合わせたクラスライブラリの作成。
- 動的 QOS 制御を、以下 2 つの性格をもつものに分離して、それらの間を時間的に遷移するようなモデルを考えてみたい。
  - Admission 制御での資源見積もりが不正確なとき、それを実勢値に収束させるための制御
  - システムの一過性の過負荷状態に対応するための制御

  現在は本来上記の異なった性格をもつ処理を一つの機構で解決しようとしているところに問題があると思われる。

6 謝辞

本研究を行なうにあたり御協力頂いた開放型基盤ソフトウェア研究開発評価事業「マルチメディア統合環境基盤ソフトウェア」プロジェクトの皆様に感謝致します。さらに，御指導頂いている慶應義塾大学環境情報学部の斎藤信男教授、徳田英幸助教授、萩野達也助教授に感謝致します。

参考文献

[1] H. Tokuda, T. Nakajima and P. Rao. "Real-Time Mach:Towards a Predictable Real-Time System". *Proceedings of USENIX Mach Workshop*, October, 1990.

[2] 平林, 斉藤, 野村, 山岸, 萩野. "実時間メディアサーバの設計", 第 *5* 回コンピュータシステム・シンポジウム論文集, *p25-32*, October, 1993.

[3] T. Nakajima, T. Kitayama and H. Tokuda. "Experiments with Real-Time Servers in Real-Time Mach". *3rdUSENIX Mach Symposium*, 1993.

[4] 多田, 西尾, 藤井, 堀切, 小野, 斎藤. "実時間カーネルを用いた連続メディアベースの設計". 第 *5* 回コンピュータシステム・シンポジウム論文集, *p33-40*. October 1993.

[5] John Allen Smith. "The Multi-Threaded X Server". *6th Annual X Technical Conference THE X RESOURCE 1,p73-89*. Winter 1992

[6] H. Tokuda, Y. Tobe, S.T.-C. Chou and J.M.F. Moura. "Continuous Media Communication with Dynamic QOS Control Using ARTS with an FDDI Network". *ACM SIGCOMM '92, p88-98*. 1992

[7] Apple Computer, Inc. "Inside Macintosh: QuickTime". Addison-Wesley. 1993

[8] H. Tokuda and T. Kitayama. "Dynamic QOS Control based on Real-Time Threads". *Proc. of the 4th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video, p113-122*. 1993

[9] S. Oikawa and H. Tokuda. "User-Level Real-Time Threads: An Approach Towards High Performancee Multimedia Threads". *Proc. of the 4th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video, p61-71*. 1993

[10] 河内谷, 緒方, 徳田. "連続メディア処理のためのリアルタイムスレッドモデルの拡張". 第 *48* 回情報処理学会全国大会論文集, *1H-1*. Spring 1994