システムソフトウェアと
オペレーティング・システム 69−10
（１９９５．６．２３）

連続メディア処理のためのミドルウェアの性能評価†

西尾 信彦　徳田 英幸

慶應義塾大学環境情報学部

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のミドルウェアのモデルとして Conductor/Performer モデルを提案し，それを実装している．これは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) 制御を備えたサーバ群を協調的に動作させるものである．このミドルウェアは IBM-PC/AT 互換機上の Real-Time Mach3.0 にリアルタイムスレッドを用いてマルチスレッドマルチセッションのサーバ群として構築した．また，サーバ間のメディアデータの通信のために共有メモリを用いた Cyclic Shared Buffer 機構を構築している．それらの性能評価について報告する。

キーワード　リアルタイム, マイクロカーネル, ミドルウェア, QOS 制御, 連続メディア, 分散システム

Performance Evaluation of a Middle Ware for Continuous Media Processing

NISHIO, Nobuhiko　Hideyuki Tokuda

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

Abstract

We have been developing middle ware which process continuous media in distributed environment on the basis of our Conductor/Performer architecture. We design this architecture in order to provide with media stream synchronization, shared buffer management and dynamic QOS – quality of service – control. We implement these servers on Realtime Mach3.0 on the IBM-PC/AT compatible, using its realtime thread as multi-threaded and multi-session servers. Besides, we developed Cyclic Shared Buffer mechanism for media data transmission between servers. This article reports their implementation and some performance evaluation.

Keyword　Real-Time Processing, Micro-kernel, Middle ware, QOS Control, 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．

1 はじめに

我々は，動画や音声などの連続メディアを扱うための基盤ソフトウェアの研究としてKeio-MMPプロジェクトをすすめている[1, 2, 3]．このプロジェクトでは，連続メディア処理にサービスの質(Quality of Service; QOS)の概念を取り入れ，それを制御するために計算機資源の管理を行ないシステムの挙動をできるだけ予測可能になるように努めている．

これまでに我々はマイクロカーネル上で連続メディア処理を行なうための処理モデルとして Conductor/Performer モデルを提案し[4, 5]，それを採用したミドルウェア(システムサーバ群)を現在，Real-Time Mach 3.0 マイクロカーネル[6]上に構築している[7, 9]．これは，ひとつの同期サーバ(conductor)がファイルサーバやウィンドウサーバなどの複数の各種メディアやそれに適した処理専用のサーバ群(performers)を指揮して連続メディアのストリームを構成するものである．本モデルでは conductor はまさに**指揮者**として「指揮棒をふる」ことにより適切なテンポを伝えるのみで，実際に「音楽を奏でる」のは**演奏者** (performers) であるというアナロジに基づいている．連続メディア処理を必要とするアプリケーションは conductor の与える API によりセッションを生成し，その中に各 performer を表現するメディアオブジェクトを生成し，それらを各アプリケーションの要求に従って連結してメディアのストリームを構成することによりさまざまな種類のアプリケーションを作成することが可能になることを目指している．

我々は既にこの Conductor/Performer モデルに基づいたプロトタイプシステムとして QuickTime 形式の動画ファイルの簡易再生システムを構築し，複数の performer を conductor を通して協調動作させる実験を行なっている[4]．またその後，システムを本格的に構築するためにQOS 制御機構をクラスとオブジェクトにより表現し実装することをすすめてきた[8]．その中で我々は，本モデルにおけるメディアストリームを conductor が上流から下流の performer まで順にメディアデータをリマップしていく同期ストリームと performer 間に共有メモリを用意して非同期に駆動する非同期ストリームとで構成している．非同期ストリームのメディアデータの受け渡しのための共有メモリ機構として Cyclic Shared Buffer(以下CSB) を考案した．これにより複数サーバによるモデルを実装したシステムが性能を落さないように注意を払ってきている．本稿では，Conductor/Performer モデルに基づいた連続メディア処理のためのミドルウェアシステムの実装を，CSB 機構を用いることによりどのように柔軟性と可用性を広めることができたかを説明し，基本的な performer 群を用いたミドルウェアシステムに基本的な性能評価について報告する．

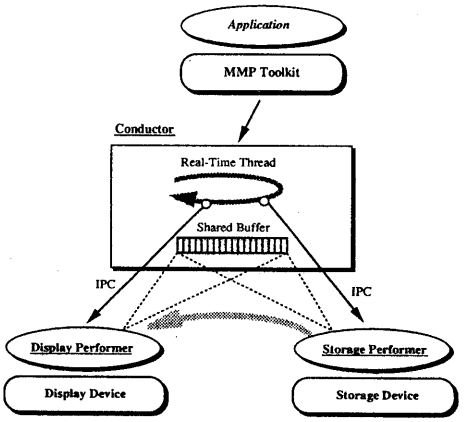

図 1: Conductor/Performer モデル

2 Conductor/Performer モデル

2.1 Conductor の概要

Conductor は各ホストに一つづつ存在し，そのホスト内の資源を管理する．その処理は大きく3つのインタフェースに分けて考えることができる．

第一にアプリケーションとのインタフェースとして，セッション[1]の生成，ストリーム[2]の構築，ストリームの動作制御 (play, stop, rewind, cue, etc.) や，QOS の変更を受け付ける．通常，連続メディア処理ではストリーム単位に処理の起動周期が決定されるので，1 つのストリームにつき 1 つの周期スレッドを conductor タスク内に生成し，メディア再生の実時間性を保証する．またスレッドのデッドラインミスにより起動するデッドラインハンドラなどを動的 QOS 制御のトリガとする．

第二のインタフェースはそのホスト内の performer とのものである．Conductor は実際にはメディアデータをアクセスすることはなく，それを代行する各 performer に対してストリーム駆動のための同期信号のみを発信する．また performer への QOS 変更要求の発行も司る．このインタフェースに関しては次の performer の項で詳しく説明する．

最後はリモートホストの conductor とのインタフェースで，リモートホストに接続したビデオカメラからの映

[1]例えば動画と音声のように互いに同期することに意味のあるストリームの集合のこと．
[2]分岐しない単一メディアデータの流れのこと．

像をローカルでモニタしたり，ファイルサーバがリモートにある場合などの，連続メディアの分散ストリームを構築するためのものである．この分散ストリームを構築する場合，一本のストリームでも両ホストの conductor 内にはひとつづつの周期スレッドが生成されることとなる．分散化したストリーム中のどれかの conductor から発行された QOS 制御要求もこのインタフェースを通して伝播する．

2.2 Performer の概要

Performer は連続メディアデータを実際に加工編集したり転送するといった各種メディア専用の処理をする．セッション開始前には基本的に conductor から要求された QOS 指定を保証できるだけの資源の見積りとその確保を行なう．

Performer の例としてはストアドメディアのためのファイルサービスや X Window を拡張したウィンドウサービス，ビデオやオーディオなどの入出力のための専用ボードを制御管理するサーバなどの他，データの圧縮伸張や動画像のエッジ認識や音声データの周波数解析からカラオケのためのボイスキャンセリングなどのソフトウェアによるメディアデータへのフィルタリングをするためのサーバも含めて考えている．

実際にアプリケーションが conductor を通して performer をつないで連続メディアのストリームを作った例を図2に示した．これは，Source Obj A がオーディオキャプチャ (マイク) で歌い手から音声データを，Source Obj B がカラオケサーバでオーケストラ音声データと動画 (歌詞付き) データをフェッチし，Filter Obj A が歌声にエコー効果を入れ，Filter Obj B, C が圧縮されていたストアドメディアを伸張し，Filter Obj D で 2 つの音声データを同期させながらミキシングし[3]，Sink Obj A で音声出力 (スピーカ) と，Sink Obj B で動画出力 (ディスプレイ) とを同期して出力させる例である．この場合，3 つのストリーム (周期スレッド) が異なった周期を持ちながら，2 箇所で同期しながら駆動される．

2.3 同期ストリームと非同期ストリーム

Conductor のクライアントアプリケーションからの視点では，conductor はメディアデータストリームの上流から下流に向けて performer を順々に連結させていくことによりストリームを構成するように単純に見せている．このとき，動画などの毎フレームを上流から下流の performer に順々にマップする処理が conductor からの同期信号によりその都度トリガされる同期ストリームと，最初にフレームレートなどの QOS のパラメータを与えておき，conductor は開始の合図のみ発行しメディアデータの転送中は同期信号を与えず，performer 間に用意された共有バッファを利用して conductor と非同期に駆動する非同期ストリームとを実装している．

[3] これはひとつのメディアオブジェクトであるが異なったストリームに含まれるので同期指定が可能となる．

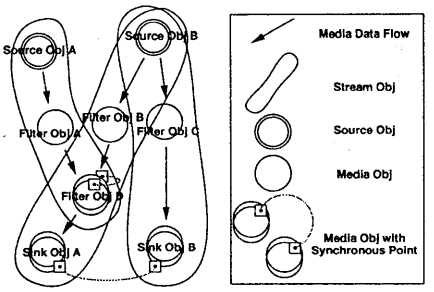

図 2: ストリーム概念図

ハードディスクへのアクセスなどのようにタイムスタンプで指定されたフレームの転送を準備する処理に遅延があったり，ネットワーク経由で転送されてくるデータのように，その時間が予測不可能である (遅延分散が大きい) 場合は非同期ストリームを使うことにより先行制御や並列制御を実現し，その遅延やジッタ傾向を吸収することを可能にすることが目的である．

この同期ストリームと非同期ストリームを順につなげてストリームを構成するが，実際に可能な単一ホスト内での流れとして以下のような場合が想定される．

同期 もっとも単純なケース．

非同期 分散ストリームの中継ホストなどに適用．

同期→非同期 下流の非同期が多段のときに適用．

非同期→同期 ディスクからやネットワーク越しなどからのストリームに適用．

同期→非同期→同期 マルチプロセッサシステムで適用．

非同期→同期→非同期 あまり適用されない．

これらのうち同期と非同期→同期が特に単一ホストでの連続メディア処理には重要となる．

いずれのストリームを用いる場合も conductor と performer との間では IPC を行なうスタブモジュールがセッション毎に生成されている．同期ストリームを構成する場合，conductor 内の周期スレッドは周期起床する毎にこのスタブモジュールを上流から下流に向けて順にトリガーしていく．この IPC し合う 2 つのスタブモジュールを以後メディアオブジェクトと呼ぶ．メディアオブジェクトは

conductor 内の新しいクラスの作成者や，conductor の API を用いるアプリケーションの作成者への programming abstraction となることを目的として導入した．

3 ミドルウェアの実装

3.1 Conductor の実装

Conductor は現在リアルタイムスレッドを用いたマルチスレッドサーバとして構成されており，C++ 言語を用いて記述されている．サーバ自体はコアとなるクラスのみで約 3000 行で記述されている．様々な performer や QOS のクラスをサポートしていくことによりこれは増えていく予定である．以下に conductor に要求できる基本的なサービスを挙げた．

- **conductor_open_session**
  (コンダクタポート，&セッションポート)
  セッション (およびセッションスレッド) の生成．
- **session_create_stream**
  (セッションポート，&ストリームポート)
  ストリームの生成．
- **stream_set_qos**
  (セッションポート，ストリームポート，QOS 仕様)
  ストリームの QOS の指定．
- **session_create_mobj**
  (セッションポート，&mobj ポート，パフォーマ ID，リソース ID)
  メディアオブジェクトの生成．
- **stream_joint_mobj**
  (セッションポート，ストリームポート，mobj ポート，接続方法)
  メディアオブジェクトの連結．
- **session_start**
  (セッションポート)
  セッションの起動 (各ストリームスレッドの生成および起動)[4]，そのセッション内の各ストリームの中の各メディアオブジェクト (パフォーマ) についてパフォーマ内処理スレッドの生成および起動．

Conductor 内にはアプリケーションからのセッション生成要求に答える大代表としてのスレッドがあり，これによりアプリケーションからの要求を受けつけて，アクティブオブジェクト (スレッドつきのオブジェクト) としてセッションを生成する．今後このセッションにおける通信はこのオブジェクト (内のスレッド) と行なう．この大代表のスレッドは通常のスレッドで，セッションスレッドからが適切な優先度の与えられたリアルタイムスレッドとしている[5]．

以後，アプリケーションからの要求でストリームとメディアオブジェクトと QOS オブジェクトが生成される．このうちストリームオブジェクトもまたアクティブで，このスレッドがストリームの実時間性を保証するための周期スレッドである．このスレッドは起床する毎にストリームの上流メディアオブジェクトから下流の順で performer に対しての IPC を行なうメソッドを起動していく．ただし，このスレッドはアプリケーションから実際に開始の合図があるまでは起動しない．QOS オブジェクトはセッション，ストリーム，メディアオブジェクト毎に生成し，アプリケーションの希望する QOS とその運用のためのポリシーを実例化している．

セッション内の異なったストリーム中に含まれる任意の 2 つのメディアオブジェクトを選んでストリーム間同期指定が可能である[6]．ストリーム内を流れるタイムスタンプのついたフレームデータはこの同期指定位置での待合わせが保証される．

まだ非同期ストリームがサポートされていなかったときには，performer 間のメディアデータの転送は Mach の仮想記憶機構によりメモリオブジェクトの vm_map により実装していた．しかし，非同期な通信のために performer 間に共有メモリによるバッファスペースを確保する必要が出てきた．そのために **Cyclic Shared Buffer (CSB)** と呼ばれる共有バッファ機構を導入した．CSB は通信しあう performer 間で共有される物理メモリである．Conductor はメディアオブジェクトを生成するときに，実際の処理を担当する performer と通信して，必要な量だけのメモリを確保し，それをそれぞれの performer のアドレス空間にマップしておく．各 performer はこのバッファとのデータの送受信と，空きメモリの送受信のために通信ポートを計 4 つ用意し，それらのポートを介した通信のみによりメディアデータが上流から下流に流せるように設計した．現在では，この CSB 機構を用いて同期ストリームも統一的に実現できるようにしてある．これに関しては performer の実装の項で述べる．

CSB 内は，動画のフレームなどの単位でアクセスするようになっていて，それ単位に所有 performer，処理状態，タイムスタンプの 3 つがマーキングされている．すべての CSB は conductor が獲得してそれを各 performer にマップするので，conductor にはすべての CSB 中のフレーム単位の処理の状態を把握することが可能になっている．

[4] 他に stop，pause，resume がある．

[5] このようにするのはリアルタイム IPC を行なうためでもある．

[6] 選び方によってはデッドロックになるので，注意が必要である．

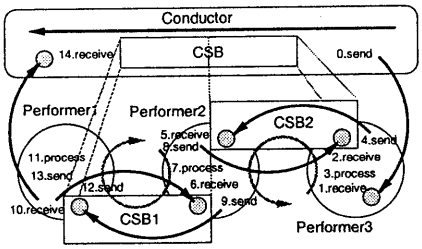

図 3: CSB(同期)

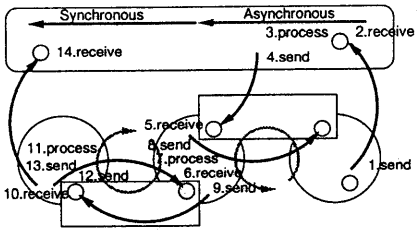

図 4: CSB(非同期→同期)

3.2 Performer の実装

Performer も conductor と同様にマルチセッションに対応できるようマルチスレッドで構成され，実時間性の保証のため RT-Mach3.0 のリアルタイムスレッドを採用して実装している．クライアント (通常は conductor) からの要求によりセッションスレッドを生成するところまでは conductor とほぼ同じである．以下に，基本的な performer の提供するサービスを挙げた．

- **performer_open_session**
  (パフォーマポート，QOS 仕様，&タスクポート，&セッションポート，
  &解放元ポート，&転送元ポート，&要求リソース)
  パフォーマのセッション (およびセッションスレッド) の生成

- **performer_bind_port**
  (セッションポート，解放先ポート)，
  転送先ポート，マップ情報)
  パフォーマの CSB 通信ポートの束縛

- **performer_session_start**
  (セッションポート)
  パフォーマ内処理スレッドの生成および起動

基本的に，performer にも大代表となる (ノーマルな) スレッドが一つあって，それとクライアントはそれと接続することによりセッションを performer 内に生成する．このときセッション毎にひとつセッションスレッド (リアルタイムスレッド) が生成され，以降クライアントはこのセッションスレッドと通信するようになる．このときクライアントは performer 毎に要求する QOS を指定する．Performer はサービスの返り引数でその QOS を保証するのに必要な資源を確保してもらうように指定する．前述の CSB のバッファの容量などはここで渡されることになる．

メディアストリームを駆動しているときの典型的な performer の処理としては，上流の performer から渡されたメディアデータにその performer 固有の何らかの処理を施し，その結果を下流の performer に渡すことである．このような動作をする performer はフィルタ performer と呼ばれ，JPEG 圧縮されたデータのソフトウェア伸張を行なう performer はその典型である[7]．

さらにこの処理を詳しく見ると，performer はストリーム内で自分の接続している上流と下流の二つの performer との間それぞれに CSB を持つことになる．よって，performer の処理は，上流の CSB からメディアデータを受けとり，下流との CSB からフリーの作業領域を確保し，そこへ加工編集結果を書き出し，それが終ると上流からのメディアデータを上流との CSB に解放して，また上流からメディアデータが渡されるのを待つことを繰り返す．

そこで我々は performer の処理は 4 つのポートとの送受信を繰り返すループにより記述できることに着目して，そのポートをどのように設定するかによって同期ストリームと非同期ストリームを performer 自身が区別することなく構成できるようにした．つまり，performer があるポートの受信待ちになっているところへ conductor からの送信によりトリガーされるようになっていれば同期ストリームをなし，上流の performer などの conductor を介さない場合は非同期ストリームを作り出せる．よってストリームの事情に合わせて conductor が 4 つのポートを適切に設定するだけで，performer 自身はストリームの接続の仕方によらず，同一の処理で対応できるようになった．

さらに，同期ストリームを形づくるときに，conductor が上流から下流に並んだ performer すべてに順にトリガーをかけていくのはいかにも不経済である．つまり同期ストリームといえど，conductor はその最上流の performer に送信をして (mach_msg_send) 最下流からの受信待ちに入る (mach_msg_receive) ようにし，同期ストリーム内の performer は自分の下流の performer にトリガーをかけていけば効率が良くなる．Performer の連結が長くなれば送受信のオーバヘッドは約半分になる．

[7] これ以外の種類の performer としては最上流と最下流の performer がある．

4 ミドルウェアの性能

我々が現在この Conductor/Performer モデルに沿って開発している，具体的な performer としては，ソフトウェアにより JPEG 方式のイメージ圧縮伸張を行なう JPEG performer，ProAudioSpectrum16 オーディオボードをサポートして音声の取り込みと再生を行なうサウンド performer，X Window サーバの DDX 層上にイメージ描画のための優先パスを設けたディスプレイ performer，VisionaryCompressionSystem の JPEG 圧縮機能付きビデオキャプチャボードに対応したビデオキャプチャperformer などである．

これらは IBM-PC/AT 互換機である Gateway2000(486DX2/66MHz) に ET4000 互換の SVGA カードを搭載したプラットフォームに，Real-Time Mach3.0 MK94 と Unix サーバである UX39 を動作させ，STAT! タイマボード (分解能 250ns) を用いて測定している．リアルタイムスレッドのスケジューリングポリシーは優先度固定 (`FixedPriority`) である．

これらの performer を CSB 経由で接続し，Mach の IPC で駆動する．現在の性能では同期ストリームにおいて performer が一つ入る毎に 400 マイクロ秒の通信のオーバヘッドが観測される．すなわち，秒あたり 30 フレームのイメージ描画を考えたとすると，performer を連続メディアのストリーム中に一つ余計に入れると 1 秒のうち 12 ミリ秒づつのオーバヘッドが必要ということになる．従来の一つの performer 毎に conductor が介在する方式の同期ストリームの場合には，この倍のオーバヘッドがかかっていた．さらに前の 1 フレーム毎に `vm_map` していく方式の場合には，20KB のデータのマップに約 1 ミリ秒，これを秒間 30 フレームならば 1 秒あたり 30 ミリ秒がメディアデータの転送かかっていたことになる．

各 performer の中での処理についてもいくつか計測してみた．JPEG performer はソフトウェアでの処理であるために性能はそれほど期待できないが，標準的な処理以外に cross-block smoothing をかける高価な処理と，グレースケールに落してしまう安価な処理の選択ができるようになっており動的に QOS を変化させるのに役立つ．320x240 の JPEG 圧縮 (1/10 から 1/5 程度に圧縮) されたイメージを伸張するのに，それぞれおよそ，450 ミリ秒，550 ミリ秒，200 ミリ秒で伸張を終了する．

ディスプレイ performer は，リアルタイムスレッドにより DDX 経由でイメージ描画する機構を持ち，クライアントとの通信でも Mach の IPC を使い，クライアントのアドレス空間にあるイメージを実際にウィンドウ内に描画するまでに一度のコピーしか要さない．120x160 の 8-bit カラーのイメージをクライアントのアドレス空間から描画終了までに標準の X Window で 61.25 ミリ秒かかるところを 12.02 ミリ秒で完了する．

5 おわりに

連続メディア処理のためのミドルウェアとして，Conductor/Performer モデルに基づいてシステムを構築している．本稿では，サーバ同士が協調するための共有メモリによるメディアデータ転送機構の CyclicSharedBuffer の性能を測定し，マルチサーバ方式にしたためのオーバヘッドの一部が測定できた．

今後は，システムのメディアストリーム間同期や動的 QOS 制御の性能の評価をするつもりである．

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、萩野達也助教授に感謝致します。

参考文献

[1] 徳田, 斎藤: “マルチメディア統合環境プロジェクトにおけるリアルタイム処理技術,” 情処研報 93-ARC-99, pp. 9–15 (1993).

[2] 斎藤, 他: “マルチメディア統合環境のテストベッドとその評価,” 情処研報 93-ARC-99, pp. 17–24 (1993).

[3] 徳田, 萩野, 斎藤: “分散マルチメディア統合環境 Keio-MMP プロジェクトにおける連続メディア処理のためのソフトウェアアーキテクチャ,” 第 49 回情処全大論文集, 7R-1 pp. 3-313–3-314 (1994).

[4] 西尾: “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,” 情処研報 94-OS-63, pp. 57–64 (1994).

[5] 西尾, 多田, 徳田, 萩野, 斎藤: “Keio-MMP における Conductor/Performer アーキテクチャの協調性能評価,” 第 49 回情処全大論文集, 7R-7, pp. 3-325–3-326 (1994).

[6] H. Tokuda, T. Nakajima, and P. Rao: “Real-Time Mach: Towards a Predictable Real-Time System,” *Proc. USENIX Mach Workshop*, pp. 73–82 (1990).

[7] 西尾, 追川, 緒方, 尾上, 河内谷, 塩野崎, 南部, 持田, 和田, 徳田: “マイクロカーネルによる連続メディア処理の基盤技術,” 第 5 回コンピュータシステム・シンポジウム論文集, pp. 17–24 (1993).

[8] 西尾, 徳田, 河内谷: “Conductor/Performer モデルにおける連続メディア処理のためのクラスの実現,” 第 6 回コンピュータシステム・シンポジウム論文集, pp. 127–134 (1994).

[9] 平林, 他: “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, pp. 25–32 (1993).