QOSの3階層指定とその翻訳を用いたセッションの単純化調停方式

西尾信彦† 徳田英幸†,††

マルチメディアシステムでは，新しいセッションを生成したり，連続メディアセッションが終了するときに返還される資源の再分配にセッション間の調停方式が重要である．エンドユーザはセッションのサービスの質（Quality of Service; QOS）を何KBといった厳密なメモリサイズやネットワークバンド幅を指定したくはないが，このような定量値はアドミッション制御には欠かせない．そこで，我々は3階層化したQOS指定と層間のQOSの翻訳機構を導入してシステムが定量値を扱えるようにした．それでもセッション間の調停には複雑で，時にはNP–完全問題に相当する計算量が必要となる．そこで，QOS指定に単調性の制約をつけることによって，セッション間調停を単純化し実装運用が可能になった．本論文では，Real-Time Mach 3.0上のConductor/Performerシステムを用いて，QOSの翻訳と複数セッション間調停の実装を報告する．

Simplified Method for Session Coordination Using Three-level QOS Specification and Translation

NOBUHIKO NISHIO† and HIDEYUKI TOKUDA†,††

In continuous media systems, session coordination technique is important to accommodate a new session and re-distribute reclaimed resources when tough continuous media session is exiting. Although end-users do not want to specify the quality of service (QOS) in an exact memory size like in KB or KB/s of network/storage bandwidth, such rigid usages are necessary for admission control and session coordination. Therefore, we introduce the three-level quality specification and QOS translation mechanism to put it into the quantitative expression which the system can handle. Nonetheless, session coordination demands complicated computation or presents an NP-complete problem. Constraining QOS specification, we invent a simplified session coordination method which is feasible enough to actually implement and run. This paper reports experiments on QOS translation and multiple session coordination using the Conductor/Performer system on top of Real-Time Mach 3.0.

1. はじめに

マルチメディアシステムでは，新しいセッションを生成したり，すでに稼働しているセッションが終了するときに返還される資源の再分配にセッション間の調停方式が重要である．そのためにはサービスの質（Quality of Service; QOS）の管理が欠かせないが，すでに提案されている管理システムに決定的なものはまだ見受けられない．あるものは実装をともなわない設計のみであったり，またあるものはセッション調停のアルゴリズムが与えられていないために実装するには非常に複雑でNP–完全問題に相当する計算量を必要とする．本論文で，我々はQOS翻訳機構を備えたQOS制御ミドルウェアと動的QOS制御のための手法を説明する．これによりセッション間でQOSを定量的に交換して動的な調停が可能となる．さらにセッション調停手法は単調性の制約をQOS指定に導入することによって単純化でき，実装がより実用的になった．

2章では関連研究を眺め，3章でQOSの指定方式を導入し，4章でQOS翻訳のフレームワークを，5章でセッション間調停のための単純化した資源の動的再分配の手法について説明する．我々の実装システムはReal-Time Mach 3.0マイクロカーネルとその上で動作するConductor/Performerシステム[1),2)]と呼ばれるミドルウェアをベースとしており，4章で簡単に説明する．6章では実際の資源利用を計測したマルチメディアセッションの調停実験を報告し，7章に結論と今後の課題を述べる．

† 慶應義塾大学環境情報学部
Faculty of Environmental Information, Keio University
†† カーネギーメロン大学計算機科学部
School of Computer Science, Carnegie Melon Unviersity

2. 関連研究

QOS 制御のための資源管理方式については，Microsoft の Rialto が複数種類のシステム資源のためのQOS予約フレームワークを提案している[3]．ここでは，*resource planner*が複数セッションの資源調停を任されているが，実装についての報告はなく，QOS指定の翻訳の考え方についても触れられていない．

IBM東京基礎研究所のKawachiyaら[4]は，QOS制御の実験システムを構築し，その性能計測を行ったが，そこではプロセッササイクル資源のみの計測であった．

Tokudaら[5]の報告では，複数のマルチメディアセッションの管理を扱っている．ここでの制御機構では，各スレッドが自分で能動的に適用している．この自己安定化方式では必ずしもセッション間での要求をバランスさせることができない可能性がある．通常のマルチメディアセッションでは状態が離散的にしか安定しないので，単にビデオフレームレートを上下させるだけでは全体性能を損なうことがある．特に，あるセッションを**何時，どれだけ**向上させればよいかは困難な問題である．これに失敗するとシステムが安定しないだけではなく，システム全体に過負荷状態を引き起こすこともある．

3. QOS指定

3.1 QOS指定の特徴

挙動が予測可能なマルチメディアシステムのためには明示的なQOS指定がなされるべきである．この指定はセッション生成時のアドミッション制御や資源予約と動的QOS制御での資源の再分配に利用される．ここでのQOS指定は以下のような特徴がある：

- QOS指定は通常離散的で有界である，
- QOS指定は多次元である，
- 指定される対象には階層構造がある，
- 対象どうしにも制約があり指定が必要である，
- QOS指定の表現には複数のレベルが存在する．

最初の特徴は，主に機器やすでに記録されているデータの形式の制約により存在する．たとえば，12 fpsでのビデオ再生には意味があるが，12.3 fpsという指定は通常されない．またある機器は160×120もしくは320×240でのイメージサイズでしか撮り込めないかもしれない．QOSの指定には必ずしも連続性を期待できないことを意味し，セッション間での資源の授受に際して重要な制約となる．

次の側面のQOSの次元については，たとえば時間的な次元や空間的な次元が用いられている．時間的QOSはビデオのフレームレートや音声のサンプリングレートに関連し，空間的QOSはビデオイメージサイズや色深度のビット長や音声の量子化ビット数などに相当する．このほかに，適時性（再生遅延やジッタ範囲，ストリーム間同期の品質など）の次元を扱うことも可能である．

第3の側面である対象の階層性とは，ユーザがQOS指定をするムービーのセッションがビデオと音声のストリームから構成されているという例にみられる．またさらに，ビデオストリームはストレージアクセス要素とデータ転送要素と圧縮/伸張要素と描画要素により構成されるかもしれない．QOS指定はその対象が階層中の何なのかを明らかにしなければならない．

4番目の側面として，対象間の優先度や制約を指定することが必要である．たとえば，あるユーザにとっては音声ストリームの方がビデオストリームよりも優先度が高いかもしれないし，最前面のウィンドウを持つセッションがつねに最重要かもしれない．またユーザからの指定によらず，データストリームの最上流の要素の指定はそれ以降の下流の要素のQOSを制約することになる．

最後の側面の複数レベルの表現については次節で説明する．

3.2 階層化したQOS指定

マルチメディアシステムにおいては，アドミッション制御や動的資源再分配のためのQOS指定に複数の表現レベルがある．エンドユーザはプロセッサやメモリやネットワークバンド幅などの詳しい情報は知りたくはなく，もっと抽象的に優良可などと指定したい．ミドルウェアのレベルでは，選択されたムービーファイルやビデオカメラの機器の性能を検討し，320×240ピクセルのイメージとか15 fpsといったもっと厳密な表現を用いるべきである．一方，アドミッション制御や資源予約やその動的な再分配を実現しなければならないシステムでは，たとえばムービー再生について必要となる物理メモリやネットワークバンド幅などの資源利用の定量的な情報を獲得していなければならない．そしてさらに，その値は実行されるプラットフォームによって異ったものとなる．

ここで我々は，QOS指定の3レベル表現を導入する．それらは，ユーザレベルQOS（ULQ），ミドルウェアレベルQOS（MLQ），システムレベルQOS（SLQ）である．ULQはエンドユーザのために導入され，優良可といった抽象的な表現がなされる．MLQは連続メディアのミドルウェアレベルのソフトウェアのためで，ULQよりも具体的な，たとえば15 fpsとか16ビット

表1 3レベルのQOS指定
Table 1 Three-level QOS specification.

	Expression	User	Example
ULQ	abstract	end-user	good/fair/poor
MLQ	H/W independent	middleware	24 fps, 22 kHz
SLQ	quantitative	kernel, etc.	18%, 20 MB/s

色とか44.1 kHzステレオサンプリングなどの表現が用いられる．MLQはハードウェアプラットフォーム独立の形式で記述する．ULQとMLQとの違いは，MLQではセッションのQOSが唯一に決定するが，ULQでは決まらないことである．ULQからMLQへの翻訳は通常はミドルウェアやアプリケーションプログラムにリンクされるツールキットライブラリでなされる．

SLQはカーネルや資源マネージャやアドミッション制御部やセッション調停部で用いられる．物理メモリサイズやネットワークバンド幅といった厳密な表現が用いられ，ハードウェアプラットフォーム依存の情報を含んでいる．これらはアドミッション制御やセッション調停における資源の動的再分配で利用される．MLQとSLQでの翻訳は通常はミドルウェア（セッションサーバや資源マネージャなど）やカーネルで行われる．

表1にこれら3様式のQOS指定の特徴を示した．

3.3 QOSパス

あるセッションに単一のQOSしか指定しないと，システムはそのセッションを対して動的適応させることができず，柔軟性が低くなる．動的QOS制御や資源の再分配を効率良く行うには，1つのセッションにつき複数のQOS状態の指定を候補にあげておくことである．それぞれにQOSの指定により要求される資源の量が異なるようにしておくことによって，動的にセッション間での資源の授受ができて調停が可能になる．

そこで，我々は，ユーザが許容する複数のQOS指定を束ねる**QOSパス**の考え方を導入した．ユーザは複数のQOS指定をするが，それぞれに好ましい順序があろう．その順序が全順序関係にあるとき，指定をその順に並べたものがQOSパスである．つまり，ユーザは自分に最も好ましい指定を端点に，そこから妥協していく順に指定することになる．システムによるセッション調停は，別に与えられた各セッションの重要度と資源の使用状況に応じて，各QOSパスでのセッションの位置を決定し，QOSパスに沿った状態遷移を各セッションに要求することによって行われる．

全順序関係を持つという規定によりQOSパスと名付けたが，半順序関係であっても，いまの自分の指定位置に隣接するどのQOS指定に遷移すべきかがその時点で一意に決定できればシステムは構築可能である．ユーザの好みがフレームレートと画面の大きさとの間で動的に変化するような場合には，半順序関係を用いて実装することも可能である．

我々のシステムでは，QOSの指定は，(i) セッション間の調停はミドルウェアレベルで行われるのでMLQの表現を用いることとして，(ii) より単純化するためにセッションごとに指定することとした．セッションごと以外では，ビデオと音声で別々に指定するストリーム単位や，空間的/時間的といったQOS次元単位で行うことも考えられる．我々がセッション単位としたのは，ストリーム間や次元間での優先度をQOSパスの構造の中に包含し単純化できるためである．

以上から，QOSパスをセッションごとに指定することによって，最初にあげたQOS指定の特徴を反映してシステムを構築することが可能になった．

4. QOSの翻訳

本章では，我々のベースとするミドルウェアの簡潔な説明をはじめに与え，次にそれに与えたQOS翻訳機構について説明する．

4.1 ミドルウェアアーキテクチャ

我々のソフトウェアプラットフォームは *Conductor/Performer* アーキテクチャと呼ばれ，Real-Time Mach 3.0マイクロカーネル上に開発した複数のシステムサーバとライブラリによって構成される．

Conductorはセッション管理/資源管理/メディア同期/QOS制御に責任がある．**Performer**はメディア固有の処理，つまりファイルサービスのperformerはストレージ機器から直接メディアデータをアクセスし，ディスプレイperformerはイメージデータを受けてウィンドウにそれを描画するといった処理を行う．ユーザはconductorにセッションの生成を要求し，conductorはそのセッションに必要なperformerと折衝して適切な資源の予約を行う．マルチメディアセッションが確立すると，リアルタイム（周期）スレッドがconductor内に生成され，それが適切にperformerとIPCすることによってメディアの同期を守る．Conductorおよびperformer間のデータの転送はすべて共有メモリのバッファ（Cyclic Shared Buffer; CSB）を経由する．システムの概観を**図1**に示した．

Conductorのセッションは1つ以上の**ストリーム**から構成される．ストリームとは分岐しない単一種類のメディアの流れを表している．たとえば，ムービーセッションはビデオストリームと音声ストリームから構成される．ストリームはそれを構成するperformer

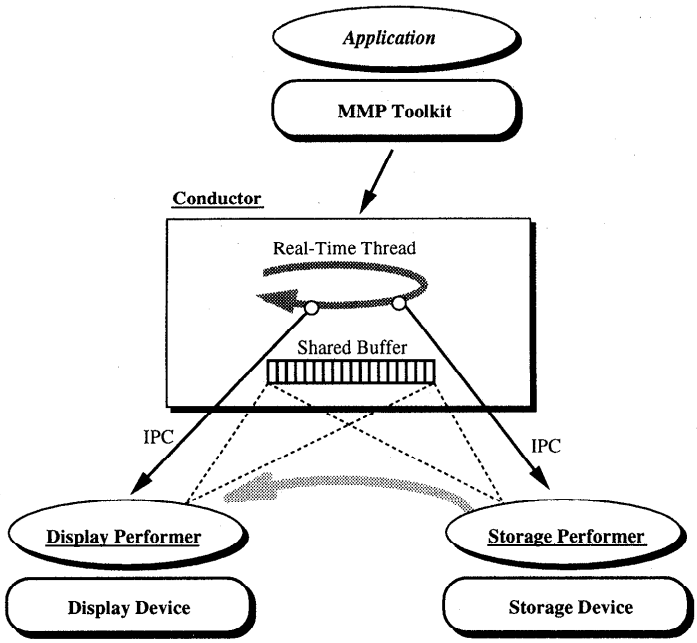


図1 ミドルウェアアーキテクチャ
Fig. 1 Middleware architecture.

の系列の抽象化である．ディスクストレージからビデオフレームをウィンドウに描画するビデオストリームはファイル performer のセッションとディスプレイ performer のセッションの系列となる．

Conductor は実際のシステム資源の予約や利用状況監視を適切な担当モジュールに委任できる．たとえばプロセッサ資源をマイクロカーネル[6]や外部の専用サーバ[4]に委ねたり，ストレージ/ネットワークのバンド幅をストレージ/ネットワーク performer[7]に委ねて構成することができる．

4.2 QOSの翻訳機構

まずはじめに，アプリケーションプログラムはGUIなどを利用してユーザにユーザレベル QOS（ULQ）を選択させる．我々のツールキットライブラリは以下の点に留意して開発されている：

- 新規セッションの優先度，
- セッションに含まれる各ストリームの優先度，
- 各ストリームの QOS 次元ごとの優先度，
- 各ストリームの QOS 指定の許容範囲．

はじめの3つは対象（セッション，ストリーム，QOS次元）間での指定である．特に，セッションの優先度は最も重要であり，これはセッション間調停での犠牲（victim）セッション選択アルゴリズムで用いられる．ストリームの優先度はどのストリームが，たとえばビデオと音声とでユーザにとって重要なのかを指定する．これらの優先度は動的に変更されても構わない．我々は以下のような動的な優先度変更のケースを想定している：

- ユーザが変更を要求するコマンドを発行した，
- 新たに生成したセッションがより高い優先度を持っていた，
- ウィンドウマネージャが最前面ウィンドウの切り替えを検出した．

4つ目の QOS 指定の許容範囲は，ULQ では抽象的に与える．我々のシステムでは **max**，**medium**，**any** といった形で与える．**Max** とは最高の QOS でしかユーザは許さない．この場合，QOS パスは最高の QOS 状態の1点のみによって構成される．**Medium** では，ユーザは中間程度の QOS の低下を許容することを意味する．**Any** はいかなる QOS をも許容するものである．

ユーザはどのストリームもその源（すなわちビデオカメラやムービーファイルの名前など）を必ず指定するので，conductor はその源を担当する performer に訊くことによって最高の QOS 値を知ることができる．Conductor は ULQ の表現をミドルウェアが扱えるより具体的な表現（MLQ）に翻訳する．最初に，源の performer にそのストリームの可能な QOS の範囲を訊く．たとえば，QuickTime ファイル performer にムービーファイル名を与えて，イメージサイズや色深

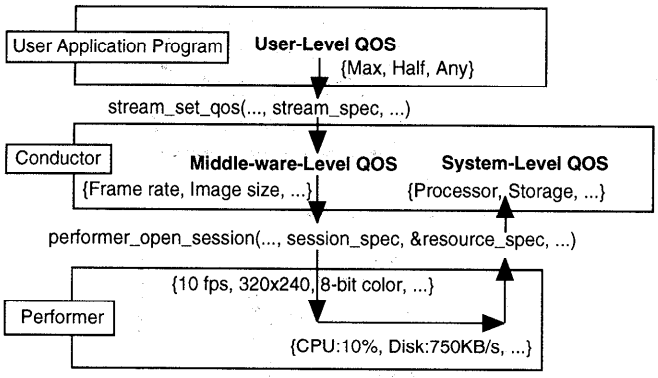


図2 QOS翻訳の流れ
Fig. 2 QOS Translation flow.

度やフレームレートなどの可能なQOS値の範囲を獲得する．このQOS値の範囲がMLQの表現として用いられる．Conductorはこれと対象（セッション，ストリーム，QOS次元）間の指定に基づいてQOSパスを適切に生成する．このQOSパスがセッションの許容されたQOS指定と遷移パスを表している．これによりULQからMLQへのQOS翻訳が行われた．

次に，MLQのQOSパスはそれが必要とする資源が実際に確保できるか調べるために各performerに渡される．というのも，その処理にかかる実際のコストはその処理を担当するperformerしか知らないためである．そこで，performerはconductorに対して，(i) 複数品質でのサービスのためのMLQ/SLQ翻訳表と，(ii) 異ったハードウェアプラットフォームのためのコスト見積りを較正する能力を提供しなければならない．

我々のシステムではMLQ/SLQ翻訳表はオフラインで生成している．各performerはその表をハードウェアプラットフォームに合わせて較正することができるようになっている．我々の用意したperformerのインタフェースでは，*class_name*_open_sessionにMLQ指定を受渡し引数で与え，返り引数で要求される資源の利用量が戻ってくる．この資源量がシステムレベルQOS（SLQ）であり，プロセッササイクルや物理メモリサイズやストレージ/ネットワークバンド幅で表現される．これで，conductorはMLQからSLQへという第2段階の翻訳を完了することができる．この段階で，conductorは新しいセッションのアドミッション制御が可能になり，セッション調停が定量的にできるようになる．ここでconductorがそのQOS範囲では調停が不可能と判断したセッションは生成されない．**図2**に我々のシステムでどのようにULQがMLQを経てSLQに翻訳されるかを示した．

5. セッション調停のための単純化アルゴリズム

Conductorのセッション調停機能は以下のタイミングで起動される：

case 1 新たなセッションの生成が要求された，

case 2 走行していたセッションが終了した，

case 3 ユーザがQOSの変更を要求したり，performerがQOSの変更要求をconductorにupcallした，

case 4 conductor内のリアルタイム（周期）スレッドのデッドラインハンドラによって集められた情報からシステムの一時的な過負荷状態が起きていると推測された．

するとconductorは以下の順番でQOS管理処理を起動する：

(1) 犠牲となるセッションの選択，

(2) そのセッションをQOSパスに沿ってQOSの状態遷移をさせたときに解放されるSLQを計算し，資源の再分配が可能かを検討，

(3) 既存のセッションのQOSの状態遷移を実行，

(4) もし必要なら，新しいセッションのために資源を予約する．

我々の資源の再分配計算は以下の制約を導入することにより単純化された．すなわち，**「QOSパスにより指定された全順序関係の順に要求される資源量が単調に減少すること」**である．このセマンティックは，「ユーザが望ましいとしたQOS指定ほど必要とされる資源量が多く，妥協していく指定の順に必要とされる資源量が減っていき，途中がその逆転があってはならない」ということである．シンタクティックには「SLQで表したQOSパス上の指定は各種資源の利用量のベクタであり，QOSパスに与えられた全順序関係で各

ベクタの要素が単調に減少している」という制約である．つまり，あるQOS指定のベクタの各要素はQOSパス上のそれより上方のベクタの要素より必ず小さいか同じとなる．

この制約による，資源再分配計算の単純化の効果は絶大である．この制約がない場合には，この計算は厳密には全解探索が必要となり膨大なものになってしまう．なぜならこの制約がなければ，QOSパス上を下方に遷移させても，ある資源の利用量は減らせるが，別の資源の利用量が増えるということが起きてしまう．つまり，パス上の下方への状態遷移が必ずしも問題解決へとは向かわず，ある新セッションのためにどのような遷移を行えばいいかは全解探索が必要となる．逆にこの制約があれば，下方への状態遷移により問題が必ず単純化することを保証できる．

では，この制約を導入することにより何を失うかを考えてみると，1つのセッションの中での使用する資源のトレードオフが指定できない点にある．たとえば，動画の圧縮/伸長をソフトウェアでやったとすると，やらない場合と比べて，データの転送量は減少するがプロセッサ資源の使用量は拡大する．このような資源使用量が単調減少しないようなQOSパスは指定できなくなる．

犠牲セッションを選択し，そのQOSパス上での遷移をすれば資源再分配計算は容易にできることが分かったので，次に我々の用いた犠牲セッションの選択法について説明する．この選択法もまた単純で，セッションにつけられた優先順位によって行われる．セッションの優先順位は既存のセッションに全順序をつけたものである．我々のconductorの実装のデフォルトでは，最も最近に投入されたセッションの順位が最も高くなる．ユーザはこれとは別に陽にセッションの順位を指定することもできる．

このようにして犠牲セッションの選定はセッションの優先順位によって，最も低いセッションが選ばれる．この犠牲セッションのQOSが1段階下げられた後，解放された資源で資源の不足が補えない場合には，2番目に優先度の低いセッションが次の犠牲セッションとなる．

最も低いセッションを2段階下げるというポリシーも考えられるが，このポリシーは優先度が高めのセッションに向くために採用しなかった．これは稼働し続けていられるセッションの数を減らして，残りのセッションのQOSを上げることになる．我々はこれを**個人主義**のポリシーと呼んでいる．我々は稼働している各々のセッションのQOSを下げて，その数を維持しようというポリシーを採用している．このポリシーを**公共の福祉**のポリシーと呼んでいる．

QOSを下げるセッションの辺りは優先度が低いので公共の福祉のポリシーが選択されているが，優先度が高いセッションでは個人主義のポリシーをとった方がいいこともある．我々はすべてのセッションを2つのグループに分けて，それぞれで資源を解放するポリシーを変えた．この2つのグループを分けるラインを**福祉の境界線**と呼ぶ．この線よりも上位のセッションは個人主義のグループに入り，残りのセッションは公共の福祉のグループに含まれる．個人主義グループのセッションは（優先順ではあるが），つねに可能な限り高位のQOSを追求し，資源をできるだけ獲得しようとする．これらのセッションは自分より下位のセッションの存続には頓着しない．これによって継続できなくなる下位のセッションも生じうる．公共の福祉グループのセッションは自らのQOSを下げても存続するセッションの数を維持もしくは増やそうとする．個人主義のセッションの数はハードウェアの能力によって決定され，実際の実装では1つないし，なし（すなわち，この場合はすべてが公共の福祉グループとなる）とした．

図3にはセッション（内のストリーム）の優先順位とQOSパスの関係を図示した．

6. マルチメディアセッション調停の実験

実験では東芝のPentium 90 MHzのサブノート（メモリ16 MB，C&T65546ビデオチップ，VLバスアーキテクチャ）をハードウェアプラットフォームとした．この実験のために2つのperformer，QuickTime File Performer（qtfp）とMMP-X Display Performer（mmpx）を用意した．

QtfpではCRAS[9]のような定レートストレージアクセスサーバを想定しており，メモリマップトファイルを用いてそれをシミュレートしている．performerは多くの物理メモリを消費するが，これによって挙動が予測可能になり，我々の実験には向いている．QtfpはAppleのQuickTimeムービーファイル形式を読むことができ，任意のタイムスタンプのイメージフレームを一定の資源コストでperformerインタフェースに則して取得することができる．本実験では，50フレームの8ビット色で160×120と320×240の2種類のビデオトラックを持つファイルを用意した．Qtfpはリアルタイム周期スレッド（もしくはプロセッサ資源予約機構）を用いて，データの転送レートの指定して上限をつけることができる．**図4**にシステムの概要を

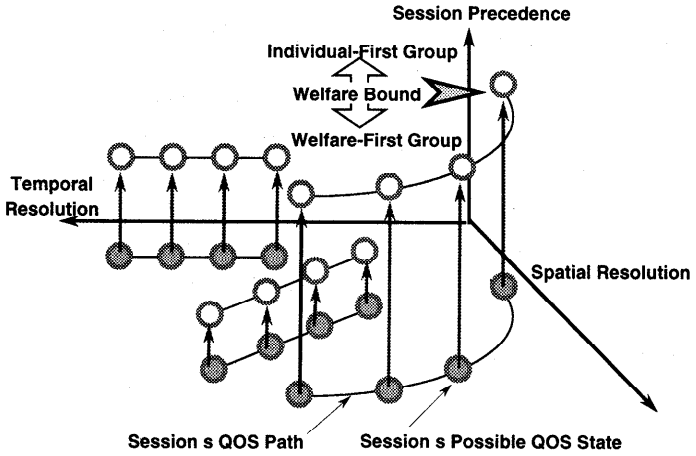


図3 QOSパスと優先順位
Fig. 3 QOS path and precedence.

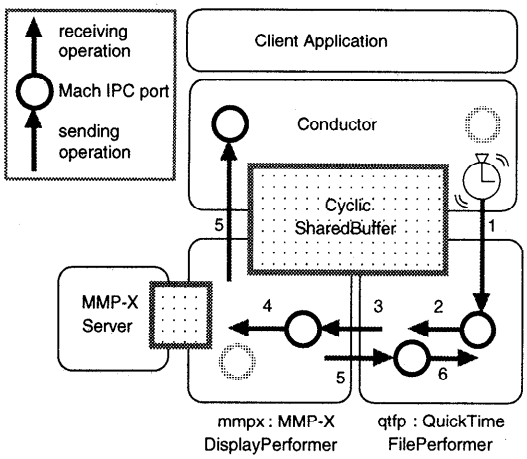


図4 実験システムの概要
Fig. 4 Experiment System Overview.

表2 QOS指定：ユーザレベルQOS表現
Table 2 Session specification: User-level QOS expression.

	Temporal Res.	Spatial Res.
Session A	Any	Max
Session B	Max	Any
Session C	Medium	Medium

表3 QOS指定：ミドルウェアレベルQOS表現
Table 3 Session specification: Middle-ware-level QOS expression.

	Frame Rate	Image Size
Session A	1, 2, 3, 5, 6, 10, 15, 30	320 × 240
Session B	30	160 × 120, 320 × 240
Session C	15, 30	160 × 120, 320 × 240

示した．

Mmpx は XFree86 v.3.1.2 の X サーバを拡張した performer である．これには，Mach-IPC のポートをリアルタイムサービス専用に通常のソケットインタフェースとは別に用意してある[10]．このサービスポートを経由した要求はソケットからの要求よりも優先して実行され，かつ共有メモリを用いてコピーの回数を減らして転送できるために高速のイメージ描画が可能となっている．

時間計測はすべて Pentium のサイクルカウンタレジスタを読みプロセッサの周波数で較正して行っている．

本実験では QuickTime ファイルを X ウィンドウに**表2**にあげた ULQ で描画する3つのセッションを用いている．これらの QOS 指定は，**表3**に示した MLQ 表現に翻訳される．表2には表れていないがセッション C は時間的 QOS が空間的 QOS よりも重要だと指定されている．

この翻訳の結果，3つのセッションのために3つの QOS パスが生成される．セッション A の QOS パスは (フレームレート, イメージサイズ) とすると，$\{(30, 320\times240), (15, 320\times240), \ldots, (1, 320\times240)\}$ である．B は $\{(30, 320\times240), (30, 160\times120)\}$ である．C は $\{(30, 320\times240), (30, 160\times120), (15, 160\times120)\}$ である．

本プラットフォームでは，qtfp performer はほとんどプロセッササイクルを消費せず，mmpx performer では 160 × 120 のイメージの描画に 1,666 マイクロ秒消費した．Conductor 内での IPC によるオーバヘッドは1フレームあたり 638 マイクロ秒で，これはイメージの大きさに関係しない．Qtfp のストレージアク

表4 QOS指定：システムレベルQOS表現
Table 4 Session specification: System-level QOS expression.

	Processor Cyc.		Storage Bandwidth	
Frame Rate	320 × 240	160 × 120	320 × 240	160 × 120
30 fps	21.9%	6.9%	2,250 KB/s	562.5 KB/s
15 fps	11.0%	3.5%	1,125 KB/s	281.25 KB/s
10 fps	7.3%	2.3%	750 KB/s	187.5 KB/s
6 fps	4.4%	1.4%	450 KB/s	112.5 KB/s
5 fps	3.7%	1.2%	375 KB/s	93.75 KB/s
3 fps	2.2%	0.7%	225 KB/s	56.25 KB/s
2 fps	1.5%	0.5%	150 KB/s	37.5 KB/s
1 fps	0.7%	0.2%	75 KB/s	18.75 KB/s

セスバンド幅には2,500 KB/sの制限を加えた．**表4**に各品質で使用される資源量を示した．

本実験では，最上位のセッションのみが個人主義グループで，セッションA，B，Cはともに最上QOSで21.9%のプロセッササイクルと2,250 KB/sのストレージバンド幅を消費する．どちらの資源も1つのセッションであれば賄うことができる．もし，A，BそしてCの順にセッションを生成したなら，Aは最上のQOSで始まり，Bが始まるとと BはAよりも重要でストレージバンド幅が不足しているので，Aは自分のフレームレートを3 fpsにカットする．そこにCが始まると，Cの最上のQOSはBの走行を止めてしまうが，Aは3 fpsのまま継続する．

個人主義グループをなくした場合には，Cが始まるまでは変化はない．Cが始まったときに，もしCが最上で走行すればBはCを走らせるために終了しなければならない．そこで，Cはイメージサイズを160×120にカットして，Bもストレージアクセスを562.5 KB/sまでカットし，Aはそのおかげで15 fpsで走行できる．

Mmpxは160 × 120のイメージをソフトウェアでコピーすることにより320×240に拡大する機能があるが，これには余計にプロセッササイクルを消費してしまう．我々のシステムではこれが1フレームあたり26ミリ秒かかった．このようなソフトウェアによるイメージ拡大は選択肢としては望ましくはあるが，我々のQOSパスの単調性の制約に反しているために採用できない．すなわちプロセッササイクルが上がり，ストレージバンド幅が下がるためである．

7. 結　論

マルチメディアセッションの調停には定量的な資源管理のもとでのアドミッション制御が不可欠である．これはシステムの各要素が定量的なサービスコストを意識して設計されていなければならない．加えて，資源予約やその利用監視機構も必要となる．我々の研究はマルチメディアセッションを厳密に調停するためにそのような予測可能な環境を目的としている．本論文では，(i) QOS翻訳機構と，(ii) 単純化されたセッション調停アルゴリズムによって，より安定した予測可能なシステム資源の再分配を可能とした．またConductor/Performerシステムを用いたQOS翻訳実験を行い，定量的な多段階品質のムービー再生セッションの調停を実現した．

今後は，本システムを分散化したメディアストリームにも対応させるために，ネットワークの資源管理を実装していく予定である．分散ストリームの場合には本稿のようなエンドシステムとは異なり，データ転送の遅延が無視できない．データ転送の遅延はシステムレベルでは，ネットワークバンド幅というよりも，転送処理のためのプロセッサ資源やバッファとして用いるメモリ資源として表現される．これに適したネットワークプロトコル層や資源管理機構の実装が課題である．

謝辞 筆者らは慶應義塾大学MKngプロジェクトのメンバに，その協力と助言について深く感謝いたします．

参考文献

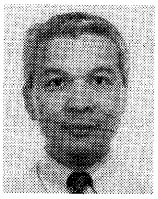
1) Keio-MMP Project: World-Wide Web Home Page, URL: <http://www.mmp.sfc.keio.ac.jp/> (1996).
2) Nishio, N., et al.: Conductor-Performer: A Middle Ware Architecture for Continuous Media Applications, *Proc. 1st Intl. Workshop on Real-Time Computing Systems and Applications*, pp.122–131 (1994).
3) Jones, M.B., et al.: Support for User-Centric Modular Real-Time Resource Management in the Rialto Operating System, *Proc. 5th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp.55–65 (1995).
4) Kawachiya, K., et al.: Evaluation of QOS-Control Servers on Real-Time Mach, *Proc. 5th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp.123–126 (1995).
5) Tokuda, H., et al.: Dynamic QOS Control based on Real-Time Mach, *Proc. 4th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp.113–122 (1993).
6) Mercer, C.W., et al.: Processor Capacity Reserves: Operating System Support for Multimedia Applications, *Proc. 1st Intl. Conf. on*

Multimedia Computing and Systems, pp.90–99 (1994).
7) Kihara, S., et al.: An Implementation of ST-II Protocol as a Uscr-Level Server on Real-Time Mach, *4th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video* (1993).
8) Kawachiya, K., et al.: Q-Thread: A New Execution Model for Dynamic QOS Control of Continuous-Media Processing, *6th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp.149–156 (1996).
9) Tezuka, H., et al.: Design and Implementation of Continuous Media Storage System on Real-Time Mach, JAIST Research Report, IS-RR-94-15S (1994).
10) Tada, S.: Real-time extension of X window system for continuous media Applications, *Proc. First International Workshop on Real-Time Computing Systems and Applications* (1994).

(平成9年 5 月15日受付)
(平成9年10月 1 日採録)

西尾 信彦（正会員）

昭和37年生．平成4年東京大学大学院理学系研究科情報科学専攻博士課程単位取得後退学．平成5年より慶應義塾大学環境情報研究所に勤務．連続メディア処理システムの研究開発に従事．平成8年より慶應義塾大学環境情報学部助手．分散リアルタイムシステム，動的QOS制御に関する研究に従事．平成6年山下記念研究賞受賞．



徳田 英幸（正会員）

昭和27年生．慶應義塾大学より工学修士．カナダ，ウォータルー大学よりPh.D.（Computer Science）．現在，慶應義塾大学環境情報学部教授，カーネギーメロン大学計算機科学部 Adjunct Associate Professor．分散リアルタイムシステム，マルチメディアシステム，通信プロトコル，超並列・超分散システム，モーバイルシステムなどの研究に従事．IEEE，ACM，日本ソフトウェア学会各会員．情報処理学会SIGOS主査，日本ソフトウェア学会 executive board member．