外付け I/O エンジン方式を用いたストリームサーバの実現

竹内 理[†] レ・モアル ダミエン[†] 坂東 忠秋[†]

高速アクセス網の普及にともない，高品質なストリームデータの有料配信サービスが注目されてきている．このサービスの実用化には，高い配信能力，配信品質の保証機能，既存の市販ストリームサーバとの機能互換性のすべてを兼ね備えたストリームサーバが必要である．本研究では，上記ストリームサーバの実現を目指し，外付け I/O エンジン方式と呼ぶストリームサーバの構成方式を新規に提案する．外付け I/O エンジン方式では，市販のストリームサーバと専用 OS 搭載サーバを連動させることで，上記要件を同時に充足するストリームサーバを少ない開発工数で実現する．さらに，Darwin ストリームサーバと，HiTactix（著者らが開発したストリームサーバ向け専用 OS）を搭載したサーバを実際に連動させ，外付け I/O エンジン方式の定量的な評価を行った．評価の結果，外付け I/O エンジン方式を用いることで，サーバに 9 K 行程度のコードを追加するだけで，機能互換性を維持しながらも，HiTactix が提供する配信性能を保持するストリームサーバが実現できること等が明らかになった．

Implementation of a Streaming Server Using External I/O Engine Architecture

Tadashi Takeuchi,[†] Damien Le Moal[†] and Tadaaki Bandoh[†]

With broadband networks expansion, charged streaming services of high quality stream data are likely to become common. In order to efficiently implement such services, streaming servers providing high data throughput performance, mechanisms to ensure streaming quality and compatibility with existing streaming servers are desirable. In this paper, we propose the external I/O engine architecture, which implements a streaming server as described above using both a conventional streaming server running on a general-purpose OS and an I/O engine server implemented over a special-purpose OS. With this combination, we can lower the necessary coding amount in order to implement the services provided. We implemented the server using the Darwin Streaming Server and HiTactix I/O engine server and evaluated the performances. We confirmed that the modified server can achieve HiTactix I/O performances without losing compatibility, with only 9 K lines of code added.

1. はじめに

近年，光ファイバ網，CATV 網等の高速アクセス網が普及しつつある．これら高速アクセス網を提供している各キャリアは，自社が提供するアクセス網への加入者のさらなる増加を促進し，かつ，基本料金以外の追加収入をもたらすサービスの提供を模索している．特に，高品質なストリームデータの有料配信サービスは，上記サービスの有力候補として注目されている[1)]．

しかしながら，高品質なストリームデータの有料配信サービスは，現在までのところ実用化に至っていない．その理由の一部として，既存の市販ストリームサーバに以下の問題があることがあげられる．

(1) 既存の市販ストリームサーバのストリーム配信性能が低いため，少ないユーザで 1 つのサーバを共有しなければならない．そのため，サービス提供コストが高くなる[2)]．

(2) 既存の市販ストリームサーバはストリーム配信の品質保証を行えない[3)]．そのため，有料サービス化が困難である（ユーザは課金されているにもかかわらず，満足いく品質でストリームデータの配信を受けられない可能性がある）．

しかし一方で，サービス普及のためには，既存の市販ストリームサーバとの機能互換性を維持し，多数のユーザが保有するストリームクライアントアプリケーションや，サービス管理者が使いなれている各種管理アプリケーション（ストリーム配信モニタ等）を従来と同様に利用可能にした方が望ましい．

著者らは次世代ストリームサーバ向け専用 OS Hi-

† 株式会社日立製作所システム開発研究所
Systems Development Laboratory, Hitachi Ltd.

Tactixの研究開発を行ってきており，HiTactixを用いることで，低オーバヘッド，かつデータ転送レートを保証したI/O実行が実現可能であることを実証してきた[4)~8)]．本研究の目的は，冒頭で述べた課題を解決するストリームサーバの実現，具体的には，既存の市販ストリームサーバとの機能互換性を維持しながらも，HiTactixのような専用OSを利用した高性能，かつ配信品質を保証したストリーム配信を少ない開発工数で実現することにある．

上記目的の実現のため本研究では，「外付けI/Oエンジン方式」という新しいストリームサーバの構成方式を提案する．「外付けI/Oエンジン方式」では，既存の市販ストリームサーバとHiTactixのような専用OSを搭載したサーバ（外付けI/Oエンジン）を連動させる．既存の市販ストリームサーバには，外付けI/Oエンジンと連動するための改変のみを加える．一方，外付けI/Oエンジンは，従来市販ストリームサーバが行っていたストリーム配信を代理実行する．このような連動により，市販ストリームサーバとの機能互換性を維持しながらも，専用OSを利用して，ストリーム配信の性能向上，および配信品質保証が実現できる．上記実現のために必要な開発工数は，既存の市販ストリームサーバの改変工数と，専用OS上で動作するストリーム配信処理の開発工数のみに抑えられる．

さらに本研究では，Darwinストリームサーバ☆を「外付けI/Oエンジン方式」に対応させるべく改変を加え，HiTactix搭載の外付けI/Oエンジンと連動させる．そして，両サーバの連動のために必要となる開発工数が大きくならないこと，および連動オーバヘッドに起因するストリーム配信性能の劣化が大きく発生せず，HiTactixが提供する配信性能を引き出せることを定量的に検証する．

以下，2章で外付けI/Oエンジン方式の概要について述べ，さらにその実現方法を検討する．次に，3章でHiTactixの概要を示し，HiTactixを利用することにより，高性能かつ配信品質を保証したストリーム配信を行う外付けI/Oエンジンの開発が容易になることを示す．さらに，4章で外付けI/Oエンジン方式に対応させるために加えたDarwinストリームサーバの改変内容について説明し，その改変のための開発工数が大きくならないことを示す．最後に，5章で外付けI/Oエンジン方式の定量的な性能評価結果について述べ，外付けI/Oエンジンとの連動オーバヘッドに起因するストリーム配信性能の劣化が大きく発生しないこと等を定量的に明らかにする．

☆ QuickTimeやMPEG4フォーマットのストリーム配信が可能なストリームサーバ．なお，QuickTimeは米国Apple Computer社の登録商標である．

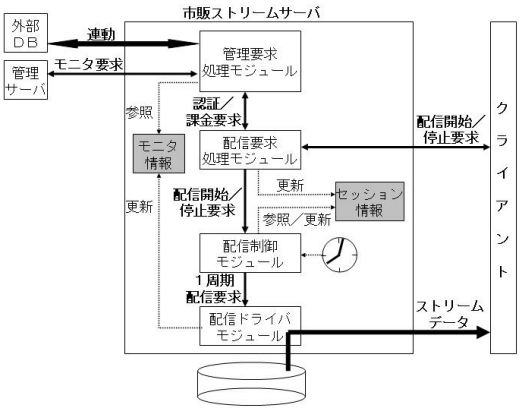

図1 市販ストリームサーバのモジュール構成
Fig. 1 Conventional streaming server modules.

2. 外付けI/Oエンジン方式

本章では，著者らが新規に提案した「外付けI/Oエンジン方式」，すなわち，既存の市販ストリームサーバにわずかな改変を加えるだけで，当該サーバのストリーム配信性能の向上，およびストリーム配信の品質保証機能の追加を可能にするストリームサーバの構成方式について述べる．

まず，市販のストリームサーバの典型的なモジュール構成について述べる．次に，外付けI/Oエンジン方式の概要について述べる．最後にその実現方法の検討結果について説明する．

2.1 市販のストリームサーバのモジュール構成

市販ストリームサーバの一般的なモジュール構成を**図1**に示す．市販ストリームサーバは，配信要求処理モジュール，管理要求処理モジュール，配信制御モジュール，配信ドライバモジュールからなる．

配信要求処理モジュールは，クライアントから配信開始/停止要求を受信する．配信開始要求受信時には，管理要求処理モジュールに対して認証/課金要求の発行を行う．また，セッション情報（現在ストリーム配信を行っているクライアントとの間で確立されているコネクションに関する情報）の更新後，当該クライアントに対する配信開始/停止要求を配信制御モジュールに対して発行する．

管理要求処理モジュールは，配信要求処理モジュールからの認証/課金要求を受信すると，必要に応じ外部データベースと連動しながら当該要求に応じた処理を実行する．また，外部の管理サーバからモニタ要求を受信すると，配信ドライバモジュールが設定するモ

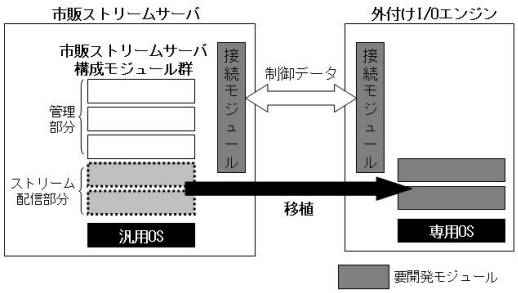

図 2 外付け I/O エンジン方式を用いたストリームサーバの構成
Fig. 2 External I/O engine architecture.

ニタ情報（CPU 負荷情報，配信レート情報等）を参照しつつ，当該情報を外部の管理サーバに返送する．

配信制御モジュールは周期的に駆動する（通常 100 ミリ秒程度の周期で駆動する）モジュールで，セッション情報に記載されているクライアントに対して，自モジュール駆動周期分の配信要求を配信ドライバモジュールに対して発行する．ストリームデータの最後まで配信が完了した等の理由で，配信すべきストリームデータが存在しない状態が一定時間以上継続している場合，セッション情報を更新する．配信要求処理モジュールは定期的に（通常 30 秒に 1 度程度）セッション情報をチェックし，上記更新処理を検知したら，当該クライアントに対する配信処理の強制終了処理，具体的には，クライアントとの間に形成されているコネクションの強制切断処理等を実行する．

配信ドライバモジュールは，配信制御モジュールから配信制御モジュール駆動周期分の配信要求を受信すると，当該周期分のストリームデータをディスクから読み込み，ネットワーク経由でクライアントに対して読み込んだデータを配信する．ストリームデータのフォーマット（QuickTime，MPEG4 等）を解釈し，当該周期に相当するデータのバイト数を決定する処理も本モジュールが行う．さらに，必要に応じてモニタ情報を更新する．

2.2 外付け I/O エンジン方式の概要

外付け I/O エンジン方式は，市販ストリームサーバの配信性能向上と配信品質保証機能の追加を，機能互換性を維持しつつも，少ない開発工数で実現するストリームサーバの構成方式である．

外付け I/O エンジン方式を用いたストリームサーバの構成を**図 2** に示す．本方式では，市販ストリームサーバと機能互換なストリームサーバを，汎用 OS を搭載した市販ストリームサーバと，HiTactix のような専用 OS を搭載した外付け I/O エンジンを連動させることにより実現する．

本方式では，市販ストリームサーバの構成モジュールを，管理部分とストリーム配信部分に分割し，専用 OS 上にはストリーム配信部分のみを移植する．さらに，両サーバ間で制御データを送受するために，接続モジュールを両サーバ上に新規実装する．

本方式では，既存の市販ストリームサーバで行っていたストリーム配信処理を外付け I/O エンジンに代理実行させる．外付け I/O エンジンは，専用 OS を利用して，高性能かつ配信品質を保証したストリーム配信を実現する．

また，本方式実現のために必要となる開発は，接続モジュールの新規実装と，ストリーム配信部分の専用 OS への移植のみに抑えられ，専用 OS 上に市販ストリームサーバの全構成モジュールを移植する従来方式と比して，開発工数の低減を期待できる．

2.3 実現方法の検討

HiTactix を用いて外付け I/O エンジンを実現する方法は，2.1 節で述べた構成モジュールのうちどのモジュールを HiTactix に移植するかにより，**図 3** に示す 3 方法が考えられる．

方法 1 は，配信要求処理モジュール，配信制御モジュール，配信ドライバモジュールを HiTactix に移植する方法である．この方法では，外付け I/O エンジンがクライアントからの配信開始/停止要求を受信し，かつ，当該要求に応じたストリーム配信を実行する．認証/課金処理や管理サーバからのモニタ要求の処理のみを市販ストリームサーバが行う．モニタ情報は外付け I/O エンジン側で更新するため，外付け I/O エンジンが保持するモニタ情報の内容を定期的（通常 1 秒に 1 度程度）に市販ストリームサーバ側に反映させ，同期をとる必要がある．

方法 2 は，配信制御モジュール，配信ドライバモジュールを HiTactix に移植する方法である．この方法では，市販ストリームサーバがクライアントからの配信開始/停止要求を受信する．当該要求を受信すると，市販ストリームサーバは外付け I/O エンジンに要求を転送する．外付け I/O エンジンは，要求に応じたストリーム配信を代理実行する．この方法では，モニタ情報以外にセッション情報も，市販ストリームサーバ，外付け I/O エンジンの双方で保持する．そのため，上記モニタ情報の同期のほかに，セッション情報の同期が必要になる．セッション情報の同期のため，定期的に（30 秒に 1 度程度），各セッションにおけるストリーム配信が正常に行われているか否かを，市販ストリームサーバから外付け I/O エンジンに問い合わせる．そして，外付け I/O エンジン側で配信完了等

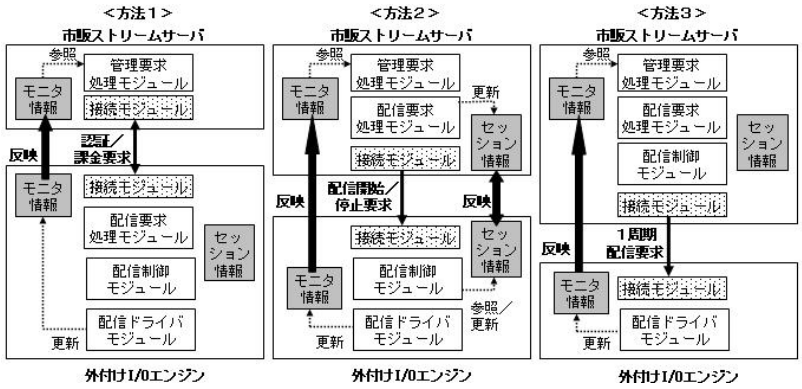

図 3 外付けI/Oエンジンの実現方法
Fig. 3 External I/O engine implementation schemes.

の理由でセッション情報の更新を行っていれば，市販ストリーム側にもその更新を反映させる．

方法3は，配信ドライバモジュールのみをHiTactixに移植する方法である．この方法では，クライアントからの配信開始/停止要求の受信のほかに，配信タイミングの制御も市販ストリームサーバが行う．外付けI/Oエンジンは，このタイミング制御に応じてストリームデータのI/O処理を実行する．モニタ情報のみ上記方法で同期をとる必要がある．

上記3方法の優劣を，以下の観点から比較した．

- 連動オーバヘッド（ストリーム配信性能劣化の可能性）
- 配信品質保証の容易性
- 既存システムへの影響
- 開発工数

比較結果を**表 1** にまとめる．

まず，連動オーバヘッドによるストリーム配信性能劣化の可能性についての比較結果を示す．連動オーバヘッドは，連動にともなう市販ストリームサーバと外付けI/Oエンジンとの間で発生しうる通信回数により比較した．発生しうる通信としては，

- モニタ情報同期のための通信
- セッション情報同期のための通信
- モジュール間での要求受け渡しのための通信

が考えられる．

先述したとおり，モニタ情報同期のための通信はどの方法でも発生し，その頻度は1秒につき1回程度である．セッション情報同期のための通信は方法2でのみ発生し，その頻度は，たかだか30秒につきN回程度（ただしNは，外付けI/Oエンジンの同時配信ストリーム数を表す）である．

モジュール間での要求受け渡しのための通信は，方法1では，配信要求処理モジュールと管理要求処理モジュール間における認証/課金処理要求にともない発生する．認証/課金処理要求はクライアントからの配信開始要求の到達にともない発生する．配信開始要求の到達頻度は，たかだかストリームデータの長さにつきN回である．この比較では，ストリームデータの平均の長さを5分（300秒）とし，たかだか300秒につきN回の頻度で発生すると仮定した．方法2では，配信要求処理モジュールと配信制御モジュール間における配信開始/停止要求にともない発生する．4章で後述するように，Darwinストリームサーバでは，配信開始/停止要求にともない6回の制御メッセージの通信を行う．すなわち，配信要求処理モジュールと配信制御モジュール間における配信開始/停止要求の通信頻度はたかだか300秒につき$6N$回である．方法3では，配信制御モジュールと配信ドライバモジュールとの間の1周期分の配信要求にともない発生する．この頻度は，たかだか100ミリ秒につきN回程度である．

以上より，1秒あたりに発生しうる最大の通信回数は，たかだか表1に記載した回数程度になると予測できる．Nが1000程度であっても，方法1，方法2では，通信回数は1秒あたり5〜55回程度で抑えられ，このオーバヘッドはストリームデータの配信にともなう通信（Ethernet経由で1.5 KBのデータ送信を1 Gbps程度の速度を行っていると仮定すると，1秒あたり80,000回以上）と比べると無視しうる．それに対して方式3では，通信回数が10,000回を超え，ストリームデータの配信にともなう通信の10%以上を占めうる．

次に，配信品質保証の容易性の比較結果を示す．一般に，QuickTimeやMPEG4等のストリームデータには配信時刻情報（タイムスタンプ）が格納されており，ストリームサーバは，この情報に厳密に従ったタイミングで配信を行う．方法1，方法2では，配信ド

表 1 外付け I/O エンジンの実現方法の比較
Table 1 External I/O engine schemes comparison.

	方法 1	方法 2	方法 3
連動オーバヘッド（通信回数/秒）	○（$1 + N/300$）	○（$1 + N/30 + N/50$）	×（$1 + N/30 + 10N$）
配信品質保証	○（HiTactix による配信制御）	○（HiTactix による配信制御）	×（汎用 OS による配信制御）
既存システムへの影響	×（サーバ IP 変更要）	○（サーバ IP 変更不要）	○（サーバ IP 変更不要）
開発工数	×	△	○

ライバモジュールの駆動を HiTactix 上に実装された配信制御モジュールが実行する．それに対して，方法 3 ではこの制御を汎用 OS 上に実装された配信制御モジュールが行う．次章で述べるとおり，HiTactix には高精度（8 ミリ秒）で配信タイミングの制御を行う GDSL（Generic Data Streaming Library）と呼ぶライブラリが存在する．一方，汎用 OS には通常このような高精度な配信タイミングの制御機能がない．すなわち，配信品質保証に関しては，方法 1，方法 2 が方法 3 より優れているといえる．

次に，既存システムへの影響についての比較結果を示す．ここでいう「既存システムへの影響」とは，市販ストリームサーバを用いて構築された配信システムに，外付け I/O エンジンを追加し，ストリーム配信性能等の増強を図る際に必要となるシステム変更の大きさを意味する．方法 1 では，クライアントからの配信開始/停止要求の送信先となるサーバと，管理サーバからのモニタ要求の送信先となるサーバが異なる．そのため，クライアントまたは管理サーバで設定されているストリームサーバの IP アドレスの変更が不可欠となる．また，外付け I/O エンジンを複数台導入した場合，クライアントにシングルサーバとして見せるためには，負荷分散装置等の導入が不可欠となる．一方，方法 2，方法 3 ではこれらの必要はない．

最後に，外付け I/O エンジンを実現するために必要となる開発工数についての比較結果を示す．接続モジュールの開発工数はどの方法でも大きな差はないため，HiTactix に移植するモジュール数が少ない方法 3，方法 2，方法 1 の順に開発工数は小さくなる．

以上の考察から，著者らは，方法 2 を用いて HiTactix 搭載の外付け I/O エンジンを実現することにした．開発工数についてのみ方法 3 より劣るが，HiTactix において，配信制御モジュール，配信ドライバモジュールを少ない開発工数で実装することを支援する GDSL というライブラリを提供することにより，その課題を一部解決した．GDSL の詳細については次章で述べる．

3. HiTactix の概要

本章では，HiTactix の概要について示す．HiTactix は，外付け I/O エンジンにおける高性能かつ配信品質を保証したストリーム配信を実現するために必要となる基本機構を保持する．この基本機構を応用し，2 章で述べた「配信制御モジュール」「配信ドライバモジュール」を HiTactix 上に少ない開発工数で実現可能とするために，著者らは，GDSL と呼ぶライブラリを新規に HiTactix 内部に実装した．以下，HiTactix が保持する基本機構と GDSL について説明し，HiTactix を用いることで外付け I/O エンジンの開発が容易になることを明らかにする．

3.1 基 本 機 構

HiTactix は，高性能かつ配信品質を保証したストリーム配信を実現するために必要不可欠となる以下の基本機能を持つ．

高速 I/O 機構 HiTactix は，他の高速 I/O を指向する専用 OS と同様，ゼロコピーでネットワーク I/O およびディスク I/O を実現する機構を持つ．さらに，アプリケーションが複数の非同期 I/O 要求を 1 つのシステムコールで一括して発行可能とする機能も持ち，I/O 完了通知オーバヘッドも低く抑えられる[8]．

アイソクロナススケジューラ HiTactix は，アイソクロナススケジューラと呼ぶタイマ駆動型のスケジューラを持つ．アイソクロナススケジューラは，**図 4** に示すように，周期駆動が必要なスレッドの CPU 割当て時間をあらかじめテーブルを用いて予約する．予約した時間にわたり確実に当該スレッドが CPU を占有できるように，HiTactix はカーネル内部の排他制御や割込み処理の管理を行っている[4]．

サイクリックディスクスケジューリング HiTactix は，サイクリックディスクスケジューリングと呼ぶディスクスケジューラを持ち，指定したストリームのディスク I/O レートを保証できる．この I/O

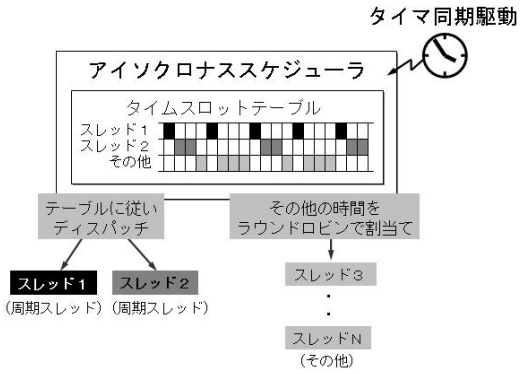

図4 アイソクロナススケジューラ
Fig. 4 Isochronous scheduler.

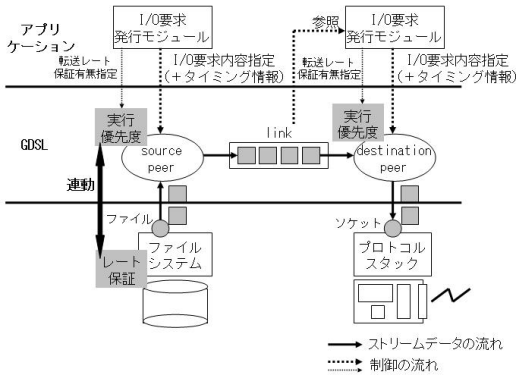

図6 GDSLを用いたストリームデータの転送処理
Fig. 6 Data streaming over GDSL.

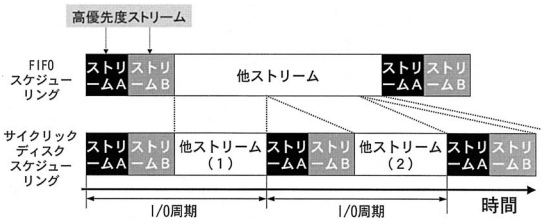

図5 サイクリックディスクスケジューラ
Fig. 5 Cyclic disk scheduler.

レート保証は，**図5**に示すように，ディスクI/Oレートを保証すべき高優先度ストリームのディスクI/Oの実行時間をあらかじめ予約し，他の低優先度ストリームのディスクI/Oによる実行遅延が大きく発生しないことを保証する．このスケジューリングを応用すると，たとえばディスクが高負荷になった場合にも，低優先度ストリームのディスクI/Oレートを落とすことで，高優先度ストリームのディスクI/Oレートを保証し続けることが可能になる[6]．

3.2 GDSL

GDSLは，2章で述べた配信制御モジュール，配信ドライバモジュールを少ない開発工数で実現することを目的にHiTactix内部に新規実装したライブラリである．以下，本ライブラリの概要と実装の詳細について説明する．

3.2.1 GDSLの概要

GDSLは，デバイス間のストリームデータ転送処理を実行する際に必要となるI/Oスケジューリング，I/O要求発行処理，バッファリング処理をアプリケーションに代わって実行するライブラリである．アプリケーションはI/Oの実行内容とその実行タイミングをGDSLに対して与える処理のみを行う．GDSLは自動的に指定されたストリームデータの転送処理を，指定されたタイミングに従い，低オーバヘッドで実行する．また，必要に応じてストリームデータの転送レート保証の有り/無しを制御することもできる．

GDSLを用いることで，高精度なタイミング制御機能を持つ「配信制御モジュール」の実現が容易になる．アプリケーションでI/Oの実行タイミング指示を与えるだけで，GDSLが自動的に高精度なI/Oスケジューリングを実現する．また，GDSLを用いることで，様々なストリームデータフォーマットや配信形態（ディスクからのオンデマンド配信，ネットワークから到達するストリームデータのライブ配信等）に対応した「配信ドライバモジュール」を少ない開発工数で実現できる．ストリームデータフォーマットや配信形態に非依存で必要となる処理はGDSLで，ストリームデータフォーマットや配信形態に依存する処理はアプリケーションで実現するようにGDSLは設計されており，「配信ドライバモジュール」開発者がこのレイヤ設計を行う必要をなくしている．

GDSLを用いたストリームデータの転送処理の概要を**図6**に示す．

アプリケーションは，GDSLの提供するsource peer/destination peer/linkと呼ぶカーネル資源を利用しながら，ディスクやネットワーク等のデバイス間におけるストリームデータ転送を実現する．source peer/destination peerはストリームデータの読み出し先/書き込み先となるファイルまたはソケットごとに生成し，当該ファイルやソケットに対するI/O要求を発行する．linkは，peer間でのストリームデータ受け渡しのためのバッファキューイング領域を保持する．

デバイス間におけるストリームデータの転送のために必要となるI/Oの内容は，アプリケーションがpeerに対応させて登録する「I/O要求発行モジュール」により制御する．I/O要求発行モジュールは，各peer

がファイルシステムやプロトコルスタックに発行すべきI/O要求の内容（I/O先となるファイル名，ファイルオフセット，送信先アドレス，I/Oサイズ，I/Oすべきデータの指定等）を指定する．destination peerに登録された「I/O要求発行モジュール」は，この要求内容を決定する際に，linkにキューイングされているストリームデータを参照することもできる．

また，peerによるI/O要求発行タイミングの制御は,「I/O要求発行モジュール」がI/O要求内容を指定する際に，あわせてI/O要求発行タイミング情報を与えることにより行う．また，アプリケーションがpeerを生成する際に，転送レート保証の有り/無しの指定を可能にすることにより，I/O要求発行の優先度の制御も可能にしている．転送レート保証有りのpeerによるI/Oの要求発行は，転送レート保証無しのpeerによるI/Oの要求発行より優先して実行される．また，ファイルに対応するpeerに転送レート保証有りを指定した場合，その読み出し/書き込みレートもサイクリックディスクスケジューリングを用いて保証される．

GDSLを用いることにより，I/O要求発行モジュールの開発のみで，HiTactix上の「配信制御モジュール」「配信ドライバモジュール」を実現可能になる.「配信ドライバモジュール」におけるサポートストリームデータフォーマットや配信形態の追加も，GDSLの提供機能を追加せずに実現できる．

3.2.2 GDSLの実装

GDSLでは，I/O要求発行モジュールからのI/O要求発行内容の受取りを，当該モジュールをアップコールすることにより実現する．また，I/O要求発行モジュールからのI/O要求発行タイミング情報の受取りは，当該モジュールのアップコールからリターンの際に，次回アップコール時刻をリターン値として受け取ることにより実現している．

GDSLでは，上記I/O要求発行モジュールのアップコールの実行，および当該モジュールから指示されたI/O要求の発行を，以下の2つのスレッドを用いて実現している．

- アイソクロナススケジューラにより，周期駆動と1周期あたりのCPU割当て時間が保証されたリアルタイムI/O実行スレッド
- インターバルタイマを用いて周期駆動する非リアルタイムI/O実行スレッド

リアルタイムI/O実行スレッドは転送レート保証有りのpeerを用いたI/O要求発行を，非リアルタイムI/O実行スレッドは転送レート保証無しのpeerを用いたI/O要求発行を行う．両者はI/O要求発行のためのCPU時間の割当て優先順位のみが異なる．両スレッドは，ともに以下に示す処理を周期的に繰り返す．

(1) 各peerに登録されているI/O要求発行モジュールを指定された時刻にアップコールする．I/O実行スレッドは，アップコール時刻管理のため，「I/O実行スケジューリングテーブル」を保持している．この詳細はすぐ後で述べる．

(2) アップコールの結果，I/O要求発行内容の指示を受け取る．また，次のアップコール時刻もあわせて受け取る．

(3) ファイルまたはソケットに対して，(2)で指示された内容に基づき，I/O要求をHiTactixの提供する非同期I/Oインタフェースを用いて一括発行する．

(4) 前の周期で発行したI/O完了をチェックする．source peerに対して発行したI/Oが完了していたら，当該I/Oの結果読み込んだストリームデータをlinkにバッファリングする．

(5) (2)で指示されたアップコール時刻情報に基づき,「I/O実行スケジューリングテーブル」を更新する．

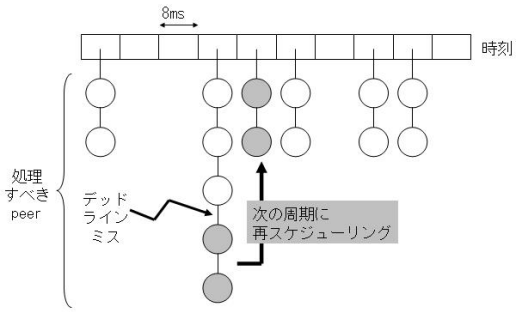

図7 I/O実行スケジューリングテーブル
Fig. 7 I/O scheduling table.

「I/O実行スケジューリングテーブル」の概要を**図7**に示す.「I/O実行スケジューリングテーブル」はI/O実行スレッドの駆動周期（現在の実装では8ミリ秒）ごとに，上記処理を行うべきpeerが登録されている．ある周期に処理すべきpeerが集中し，1周期内ですべてのpeerの処理が完了しなかった場合，残ったpeerの処理は次の周期に実行する．この結果，I/O実行要求モジュールが要求したI/O要求発行タイミングと実際のI/O要求発行タイミングとの間に誤差が発生しうる．この誤差の大きさがどの程度になるかについては，5章で定量的に評価する．

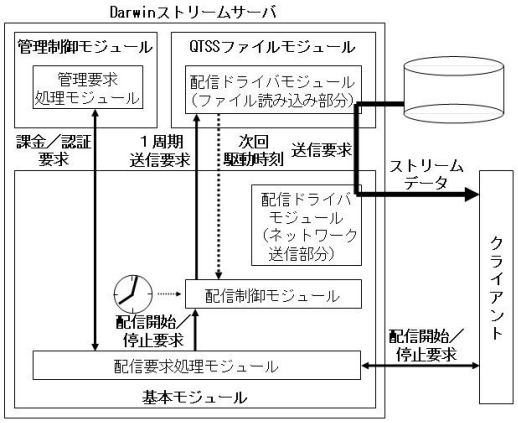

図 8 改変前の Darwin ストリームサーバのモジュール構成
Fig. 8 Original Darwin streaming server modules.

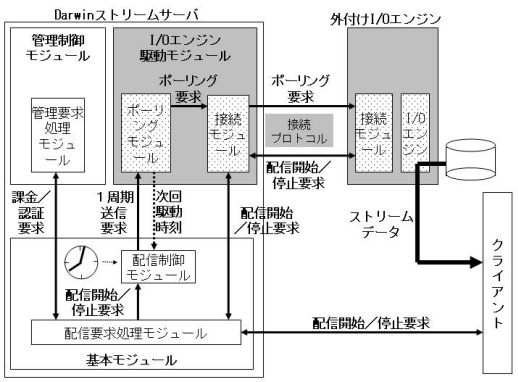

図 9 改変後の Darwin ストリームサーバのモジュール構成
Fig. 9 Modified Darwin streaming server modules.

4. Darwin ストリームサーバの改変内容

本章では，外付け I/O エンジン方式に対応すべく著者らが Darwin ストリームサーバに加えた改変内容の詳細について述べる．そして，少ないソフトウェア開発工数で，Darwin ストリームサーバの機能互換性を維持しつつ，外付け I/O エンジン方式に対応可能であることを明らかにする．

4.1 全体構成

改変前の Darwin ストリームサーバのモジュール構成を**図 8** に示す．

Darwin ストリームサーバはバージョンアップ作業を容易にするため，階層構造をしている．下位層に位置する基本モジュールは，上位層を追加実装するためのインタフェースを備えている．現在，このインタフェースを用いて，上位層に管理制御モジュールや QTSS ファイルモジュール等が実装されている．

基本モジュールには，2 章で述べたストリームサーバの構成モジュールのうち，管理要求処理モジュール，配信要求処理モジュール，配信制御モジュール，配信ドライバモジュールの一部（ネットワーク送信部分のみ）に相当する部分が含まれる．管理制御モジュールには，管理要求処理モジュール相当部分が含まれる．QTSS ファイルモジュールには配信ドライバモジュール（ファイル読み込み部分のみ）相当部分が含まれる．

配信ドライバモジュール相当部分は，基本モジュールと QTSS ファイルモジュールに分割して存在し，その詳細動作は以下のとおりになる．

(1) 配信制御モジュールが配信ドライバモジュール（ファイル読み込み部分）に対して，1 周期分のストリームデータの送信要求を発行し，1 周期分のストリームデータのファイルからの読み込みを要求する．

(2) 配信ドライバモジュール（ファイル読み込み部分）は要求されたストリームデータの読み込みを実行した後，配信ドライバモジュール（ネットワーク送信部分）に対して，当該データの送信を要求する．あわせて配信制御モジュールに対して，配信ドライバモジュール（ファイル読み込み部分）の次回駆動時刻（通常，1 周期時間，すなわち 100 ミリ秒後）を通知する．

(3) 配信ドライバモジュール（ネットワーク送信部分）は，要求されたストリームデータの送信を実行する．

(4) 配信制御モジュールが指定された駆動時刻に配信ドライバモジュール（ファイル読み込み部分）を再駆動する．(1) に戻る．

ストリームサーバは，将来的に，

- 管理機構の機能向上
- 配信可能なストリームデータフォーマットの追加
- ライブ配信サポート（サポートする配信形態の追加）

等の理由によりバージョンアップを行う☆．

Darwin ストリームサーバでは，これらのバージョンアップ時には，基本モジュールの改変はほとんど不要である．管理機構の機能向上は，管理制御モジュールの改変により実現する．配信可能なストリームデータフォーマットの追加は，QTSS ファイルモジュールに相当するモジュールを新規に追加し，新規にサポートするストリームデータフォーマットの解釈，および 1 周期分のストリームデータのファイルからの読み込みを可能にする．ライブ配信サポートは，QTSS ファ

☆ Darwin ストリームサーバがバージョン 3.3.8 からバージョン 4.0 にバージョンアップした際には，ここに示した 3 つがその主内容であった．

表 2 接続プロトコルの制御メッセージ一覧
Table 2 Control messages.

メッセージ名	対応する RTSP メッセージ	I/O エンジンへの要求内容
CreateSession	Describe	新規セッション情報の生成. ストリームデータの情報（ストリームデータフォーマット等）の返送.
SetupSession	Setup	ストリーム配信に必要なカーネル資源（peer，link 等）の生成.
StartSession	Play	ストリーム配信の開始準備 （ファイルオフセット現在値等の配信制御データ初期化）.
RunSession	Play	ストリーム配信の開始.
StopSession	Pause	ストリーム配信の停止.
DestroySession	Teardown	ストリーム配信に使用したカーネル資源の解放. セッション情報の削除.

イルモジュールに相当するモジュールを新規に追加し，ファイルからストリームデータを読み込むのではなく，ネットワークからストリームデータを読み込む．

4.2 改変内容の概要

今回の Darwin ストリームサーバの改変では，通常の Darwin ストリームサーバのバージョンアップ作業と同様，基本モジュールにはほとんど手を加えていない．**図 9** に示すとおり，上位層モジュールとして I/O エンジン駆動モジュールの新規実装を行ったのが主要な改変内容である．

I/O エンジン駆動モジュールは，接続モジュール，およびポーリングモジュールを含む．接続モジュールは，クライアントからの配信開始/停止要求の外付け I/O エンジンへの転送を行う．この転送のための接続プロトコルの詳細は次節で述べる．ポーリングモジュールはセッション情報，およびモニタ情報の同期を行うモジュールで，従来の配信ドライバモジュール（ファイル読み込み部分）を流用して実現している．

従来の配信ドライバモジュール（ファイル読み込み部分）と同様，ポーリングモジュールは配信制御モジュールから駆動される．しかし，配信ドライバモジュール（ファイル読み込み部分）とは異なり，ファイルからストリームデータを読み込まない．代わりに，セッションごとに外付け I/O エンジンが正常にストリームデータの配信を実行しているか否かのチェックを行う．このチェックは，接続モジュールに次節で述べる接続プロトコルを用いたポーリングを要求することにより実現している．正常に配信していなければ，セッション強制終了処理を実行する．配信制御モジュールに通知する次回駆動時刻を 30 秒後とすることで，ポーリングモジュールの駆動頻度を抑えている．同様の手法で，1 秒間隔でモニタ情報の同期も行っている．

外付け I/O エンジンには，接続モジュールのほかに I/O エンジンモジュールを新規実装した．I/O エンジンモジュールとは，配信制御モジュール，配信ドライバーモジュール相当部分を GDSL 上に実装したモジュールである．

以上述べたとおり，Darwin ストリームサーバを外付け I/O エンジン方式に対応させる際にも，Darwin ストリームサーバが管理するデータ構造や，QTSS ファイルモジュール以外のモジュールに改変を加える必要はほとんどなく，連動のために必要となる開発工数は大きくならない．より詳細な改変量の解析は 4.4 節で述べる．

4.3 接続プロトコルの概要

本節では，Darwin ストリームサーバと HiTactix 搭載の外付け I/O エンジンとの連動の際に使用する接続プロトコルの概要について述べる．

接続プロトコルは，

- 配信開始/停止要求を転送する機能
- セッション情報/モニタ情報の同期をとる機能

を持つ．これらの機能はすべて，Darwin ストリームサーバから外付け I/O エンジンに対して要求制御メッセージを送信し，外付け I/O エンジンが Darwin ストリームサーバに応答制御メッセージを返送することにより実現している．

配信開始/停止要求を転送する際に使用する制御メッセージの一覧を**表 2** に示す．本制御メッセージの種類は，サーバ–クライアント間で RTSP[9] プロトコルに従い送受される制御メッセージに対応して存在する．

配信要求処理モジュールは，Play 以外の RTSP 制御メッセージ（配信開始/停止要求）を受信すると，接続モジュールに対して，対応する接続プロトコルの制御メッセージの外付け I/O エンジンへの送信を要求する．接続モジュールは外付け I/O エンジンからその要求の実行結果を応答として受信する．当該実行結果は配信要求処理モジュールを介してクライアントに返送される．

配信要求処理モジュールが，Play を RTSP 制御メッセージとして受信すると，まず，StartSession の送信

表3 接続プロトコルの制御メッセージ一覧
Table 3 Control messages.

メッセージ名	I/Oエンジンへの要求内容
GetEngineStatus	モニタ情報（CPU負荷，配信レート等）の返送
GetSessionStatus	セッションの配信状態（ファイルオフセットの現在値等）の返送

表4 改変ソースコード量
Table 4 Modified source code amount.

モジュール名	改変対象量（K行，使用言語）	改変量（K行，使用言語）
接続モジュール/ ポーリングモジュール（Darwin）	1.0（C++）	3.0（C++，改変）
接続モジュール（HiTactix）	–	1.0（C，新規実装）
I/Oエンジン（HiTactix）	–	5.0（C，新規実装）
その他（Darwin）	–	0.1（C++，データ構造追加）
合計	1.0	9.1

を接続モジュールに要求する．上記と同様な手順でクライアントにStartSessionの実行結果を返送した後，接続モジュールに対してRunSessionの送信を要求し，外付けI/Oエンジンによるストリーム配信を開始させる．この制御により，クライアントへのPlay要求の実行結果の返送が，ストリーム配信開始よりも前に行われることを保証する．

Darwinストリームサーバと外付けI/Oエンジンとの間では，クライアントからの配信開始/停止要求到達にともない，6回の制御メッセージ送受信（外付けI/Oエンジンへの要求送信と外付けI/Oエンジンからの応答の受信）を行う．各制御メッセージの要求/応答には，要求/応答の種類の識別フィールド，各種要求実行時に必要となる情報（URL，クライアントIPアドレス等），実行結果が格納されている．その要求/応答のメッセージの大きさはそれぞれ最大で296バイト，272バイトである．

セッション情報やモニタ情報の同期の際に使用する制御メッセージの一覧を**表3**に示す．GetEngineStatusは1秒につき1回，GetSessionStatusは30秒につき配信実行中のストリーム数だけ，Darwinストリームサーバと外付けI/Oエンジンとの間で送受される．GetEngeineStatusにより，Darwinストリームサーバは外付けI/Oエンジンからモニタ情報（CPU負荷，配信スループット等）を取得する．現在の実装では，本制御メッセージの要求は12バイトで，応答は428バイトである．また，GetSessionStatusにより，セッションごとの配信状態（ファイルオフセットの現在値等）を取得する．本制御メッセージにより，配信が一定時間以上進行していないセッションを検知可能になる．現在の実装では，本制御メッセージの要求は12バイトで，応答は44バイトである．

本節で述べた制御メッセージの送受が，どの程度の外付けI/Oエンジンの配信性能の劣化を招くかについての定量的な評価は，5章で述べる．

4.4 改変ソースコード量の考察

本章で示した改変で，Darwinストリームサーバ/HiTactix上に新規実装，もしくは改変したソースコード量を**表4**に示す．表4に示すとおり，今回の改変は，合計で9.1K行のソースコードの新規実装，改変，データ構造の追加を行った．改変対象となったDarwinストリームサーバのコード量は1.0K行（QTSSファイルモジュールのみ）であり，Darwinストリームサーバ全体（66K行）の2%以下に抑えられた．

従来方式に従い，専用OS上にDarwinストリームサーバの全構成モジュールを移植する場合，以下の開発工数が必要になると予想できる．

(1) OS依存モジュール☆（15.0K行）のHiTactixへの移植
(2) パスワード管理機能等のHiTactixに不足している機能の新規実装
(3) 同期I/Oを用いて実現されているストリーム配信処理の高性能化

(3)の工数はHiTactix上のI/Oエンジンモジュールの新規実装とほぼ同等の工数が必要になる．(1)の工数を接続モジュール/ポーリングモジュールの実装と同等の工数であると仮定しても，(2)の工数の分だけ外付けI/Oエンジン方式は従来方式と比して開発工数を低減していると考えられる．

また，外付けI/Oエンジン方式を用いると，将来，Darwinストリームサーバがバージョンアップした場合においても，Darwinストリームサーバや HiTactix上のモジュールに大きな改変を必要としない．

☆ Darwinストリームサーバは，他OSへの移植を容易にするため，OS依存モジュールの提供インタフェースを定義し，このインタフェース上に他モジュールを構築している．

表 5 ハードウェア仕様
Table 5 Hardware specification.

サーバ種類	サーバ名	ハードウェア仕様
汎用 OS サーバ	Darwin ストリームサーバ（外付け I/O エンジン方式非適用版）	PC/AT 互換機（Pentium☆III 1.26 GHz 搭載） Gigabit Ethernet NIC（x2）（Alteon AceNIC チップ搭載） Ultra160 SCSI カード（x4）（LSILogic 53C1010 チップ搭載） Ultra160 HDD（x16）
専用 OS サーバ	簡易 Darwin ストリームサーバ	Darwin ストリームサーバ（外付け I/O エンジン方式非適用版）と同一構成
外付け I/O エンジン方式サーバ	Darwin ストリームサーバ（外付け I/O エンジン方式適用版）	PC/AT 互換機（Pentium III 1.26 GHz 搭載） Gigabit Ethernet NIC（x1）（Alteon AceNIC チップ搭載）
	外付け I/O エンジン	Darwin ストリームサーバ（外付け I/O エンジン方式非適用版）と同一構成

たとえば，管理機構の機能向上がなされても，接続モジュールや I/O エンジンモジュールの改変は不要である．サポートするストリームデータフォーマットの追加やライブ配信サポートがなされた場合には，HiTactix 上の I/O エンジンモジュールをバージョンアップする必要がある．しかし，HiTactix には GDSL が搭載されており，このバージョンアップに要する開発工数を低減できる．バージョンアップの際に追加実装すべきコード量は，たかだか現在の I/O エンジンモジュールのコード量，すなわち C 言語で 5.0 K 行程度だと予測できる．

一方，従来方式では，Darwin ストリームサーバのバージョンアップ，特に管理機構の機能向上により，HiTactix 不足機能の新規実装がさらに必要になる可能性がある．

5. 性能評価実験

本章では，外付け I/O エンジン方式の定量的な性能評価のために行った実験内容とその結果について述べる．性能評価は，配信性能，配信タイミング保証性能，配信レート保証性能の 3 つについて行った．以下，それぞれの概要を説明する．

5.1 配信性能の評価実験

配信性能の評価実験では，外付け I/O エンジン方式を用いることで，従来方式で構築した専用 OS サーバ（専用 OS 上に市販ストリームサーバ全体を移植して実現したサーバ）と比してどの程度の配信性能の劣化が発生しうるのか，また，汎用 OS サーバ（汎用 OS 上に構築されたストリームサーバ）と比してどの程度の配信性能向上が期待できるのか，定量的に評価した．

配信性能の評価実験で用いた実験システムの構成を**図 10** に示す．本実験では，汎用 OS サーバとして，改変を加えていない（外付け I/O 方式を適用していない）Darwin ストリームサーバを使用した．また，専用 OS サーバとして，HiTactix を搭載した簡易 Darwin ストリームサーバを使用した．簡易 Darwin ストリームサーバとは，外付け I/O エンジンに，Darwin ストリームサーバの配信要求処理モジュールの一部を HiTactix に追加移植することで実現したサーバである．外付け I/O エンジン方式サーバとして，前章で加えた改変を加えた（外付け I/O エンジン方式に対応した）Darwin ストリームサーバと HiTactix 搭載の外付け I/O エンジンを使用した．

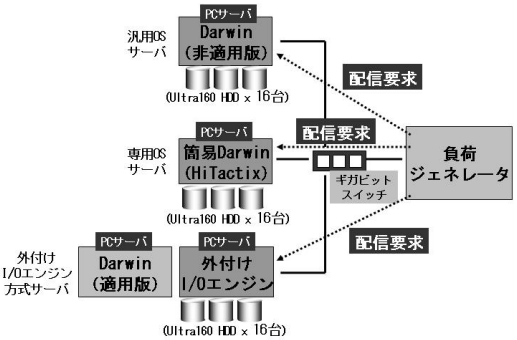

図 10 配信性能評価実験システム
Fig. 10 Experimental system for streaming performance evaluation.

本実験で用いたハードウェアの仕様を**表 5** に示す．本実験では，汎用 OS サーバ，専用 OS サーバは同一のハードウェア構成である．外付け I/O エンジン方式サーバは，上記ハードウェア構成に加えて，Darwin ストリームサーバ（外付け I/O エンジン方式適用版）用 PC サーバをもう 1 台追加してる．

実験は，負荷ジェネレータから各サーバに対して可変のストリーム数（各ストリームはビットレート 1.42 Mbps の QuickTime フォーマットのデータ）の配信要求を送信し，各サーバのストリーム配信レートおよび配信実行中の CPU 負荷がどのように変動するかを測定した．ただし，外付け I/O エンジン方式サー

☆ Pentium は米国 Intel 社の登録商標です．

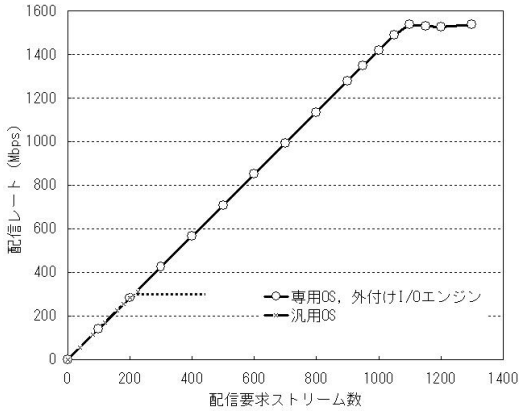

図 11 ストリーム配信レートの比較
Fig. 11 Streaming rate.

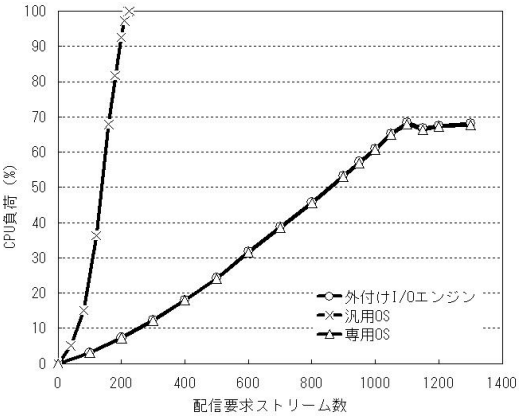

図 12 CPU 負荷の比較
Fig. 12 CPU load comparison.

バは，外付け I/O エンジンの CPU 負荷を当該サーバの CPU 負荷として測定した．配信要求は，各サーバが保持する 16 台のハードディスクから均等に配信するように発行した．また，ディスクキャッシュを使用しないように，各配信要求はすべて異なるストリームデータの配信を要求した．

測定結果を**図 11** および**図 12** に示す．各グラフの横軸は負荷ジェネレータが配信を要求したストリーム数を，縦軸は各サーバのストリーム配信レートおよび CPU 負荷を表している．汎用 OS サーバは，配信要求ストリーム数が 220 を超えると，配信の異常終了が発生する（一定時間以上配信が実行できないストリームが生じはじめる）ため，測定ができなかった．そのため，配信要求ストリーム数が 220 ストリームを超えたときは，配信要求ストリーム数が 220 ストリームのときと同じ配信レートであると仮定した（図 11 の破線部分）．また，専用 OS サーバと外付け I/O エンジン方式サーバのストリーム配信レートの変動は完全に一致するため，図 11 では両者を同一の実線で表している．

測定の結果，以下のことが明らかになった．

- 専用 OS サーバと比しても，外付け I/O エンジン方式サーバの CPU 負荷の増大は，平均で 0.49%に抑えられている．本実験では配信実行中の CPU 負荷を測定しているため，この配信性能の劣化は，2 章で述べたセッション情報の同期，モニタ情報の同期のために必要となる通信処理のみに起因している．短いストリームデータを配信した場合，配信開始/停止要求受け渡しのための通信処理による CPU 負荷増大も発生しうる．しかし，2 章で述べたとおり，5 分程度の短いストリームデータを配信した場合でも，この通信の発生回数は 1 秒あたり $N/50$ 程度（N は同時配信ストリーム数）であり，セッション情報/モニタ情報の同期のために必要となる通信の発生回数 $1 + N/30$ と比して，同程度もしくはそれ以下である．そのため，外付け I/O エンジン方式における連動オーバヘッドに起因する配信性能の劣化は 1%以下に抑えられると予想できる．
- 汎用 OS サーバは，1 台のサーバを用いて，300 Mbps 程度のストリーム配信性能を達成している．一方，外付け I/O エンジン方式サーバは，2 台のサーバを用いて 1.5 Gbps 程度のストリーム配信を達成している．すなわち，外付け I/O エンジン方式の適用による 1 サーバあたりのストリーム配信性能向上の効果は 2.5 倍程度である．なお，外付け I/O エンジン方式においては，Darwin ストリームサーバ（外付け I/O エンジン適用版）にはほとんど負荷がかからないため，同時に稼働する外付け I/O エンジン方式を増やした場合，外付け I/O エンジンの台数に応じてスケーラブルにストリーム配信性能が向上すると期待できる．このため，外付け I/O エンジンの同時稼働台数を増やすと，1 サーバあたりのストリーム配信性能の向上の効果は，さらに増大すると予想できる．

なお，専用 OS サーバや外付け I/O エンジン方式サーバの場合，CPU 負荷が 100% に満たないにもかかわらず，ストリーム配信レートが増えなくなる．これは，サーバの PCI バスの帯域が飽和しているためだと考えられ☆，さらに広帯域な PCI バスを持つサーバを使用した場合，上記の 1 サーバあたりの配信性能

☆ 実験で使用した PC サーバが保持する PCI バスの理論性能は 532 MB/秒である．1.5 Gbps のストリーム配信を行った場合，この 70%を使用している．

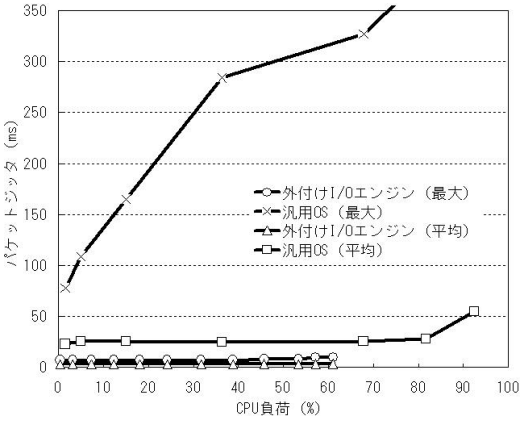

図 13 パケットジッタの比較
Fig. 13 Packet jitter comparison.

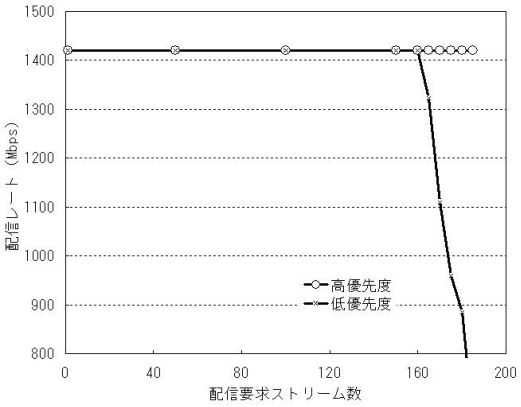

図 14 配信優先度付けの効果
Fig. 14 Prioritized streaming effects.

向上の効果はさらに増大すると期待できる．

5.2 配信タイミング保証性能の評価実験

3 章で述べたとおり，HiTactix 搭載の外付け I/O エンジンでは，GDSL を用いてストリーム配信のタイミング制御を行う．GDSL は，アプリケーション（I/O 実行要求発行モジュール）が指定する I/O 要求発行タイミングがある時刻に集中した場合，その一部の I/O 要求発行を次の周期に持ち越す可能性がある．そのため，厳密な配信タイミング保証ができない．この持ち越しによる配信タイミングの誤差がどの程度発生するか，図 10 に示した実験システムを用いて評価した．

本実験では，負荷ジェネレータから外付け I/O エンジン方式サーバに可変ストリーム数の配信要求を送信し，外付け I/O エンジンのパケットジッタがどのように変動するかを測定した．ここでいうパケットジッタとは，ストリームデータを格納する各パケットを送信すべき時刻と，実際の送信時刻との差をいう．送信すべき時刻は，QuickTime フォーマットのストリームデータに格納されている時間情報から算出でき，この時刻に基づきアプリケーションは配信タイミングを指定する．本実験では，I/O エンジンノードに 7 分間のストリームデータを配信させ，この 7 分間の最大と平均のパケットジッタを測定した．また，比較のため，同様の実験を汎用 OS サーバに対しても行った．

測定結果を**図 13** に示す．グラフの横軸は，外付け I/O エンジン，または Darwin ストリームサーバ（外付け I/O エンジン非連動版）の CPU 負荷（配信要求ストリーム数を可変にした結果変動する）を示す．縦軸は，パケットジッタの大きさを示す．

測定の結果，以下のことが明らかになった．

GDSL を用いている外付け I/O エンジン方式サーバは，少なくとも，CPU 負荷が 60%以下であるならば，パケットジッタは平均で 3.5 ミリ秒，最大でも 7 ミリ秒〜13 ミリ秒で抑えられる．現在の GDSL の実装では，8 ミリ秒を 1 周期として管理しているため，2 周期より大きい I/O 要求発行の持ち越しは発生していない．この最大パケットジッタの大きさは，汎用 OS サーバにおける平均のパケットジッタ（23 ミリ秒〜55 ミリ秒）の 1/2 以下，最大パケットジッタ（78 ミリ秒以上）の 1/11 以下である．

5.3 配信レート保証性能の評価実験

3 章で述べたとおり，GDSL は，ストリームごとにデータの転送レート保証の有り/無しを制御できる．その結果，ディスクが過負荷状態になった場合においても，優先度の低い（転送レート保証のない）ストリームのディスク読み込みレートを落とすことにより，優先度の高い（転送レート保証のある）ストリームのデータを期待されるレートでディスクから読み込み，クライアントに配信できる．

外付け I/O エンジン方式を適用した Darwin ストリームサーバが，上記に示すストリーム配信制御を行えることを確認するため，以下に示す実験を図 10 の実験システムを用いて行った．

まず負荷ジェネレータから，1 ストリームの配信要求を外付け I/O エンジン方式サーバに対して送る．この 1 ストリームは，低い優先度での配信を要求する．次に，可変ストリーム数の配信要求を負荷ジェネレータから外付け I/O エンジン方式サーバに対して送る．これらのストリームは，高い優先度で配信するように要求する．なおこれらのストリームは，優先度の低いストリームと同じハードディスクからデータを読み出すことを要求した．優先度の高い配信を要求するストリーム数を変動させた場合，各優先度のストリームの配信レートがどのように変動するかを測定した．

測定結果を**図14**に示す．図14のグラフの横軸は優先度の高い配信を要求したストリーム数を，縦軸はストリーム配信レートの平均を表す．ストリーム配信レートは，優先度の高いストリームと優先度の低いストリームのそれぞれの平均値を測定した．

測定の結果，配信要求ストリーム数が160を超えてから期待どおりに優先度の低いストリームの配信レートが落ち込むこと，および優先度の高いストリームの配信レートは要求配信ストリーム数が160を超えても厳密に一定に保てることが確認できた．

6. 関連研究

本稿で提案した外付けI/Oエンジン方式のように，市販ストリームサーバとの機能互換性維持，少ない開発工数，高性能ストリーム配信，配信品質保証のすべてを同時に実現するストリームサーバを構築しようとする研究は，著者らの知る限りでは存在しない．

従来のストリームサーバは，

- 汎用OS上にストリームサーバを構築する．ストリーム配信性能を向上させる必要がある場合には，汎用OSとインタフェースの互換性を維持しながらもI/O性能の向上をはかる機能を使用する．
- 専用OS上にストリームサーバを構築する．専用OS内部に，高性能ストリーム配信機能や配信品質保証機能を組み込む．

のどちらかのアプローチで構築されていた．

前者のアプローチとしては，たとえばLinux等の汎用OSと，Zero-Copy TCP[10]やI/O-Lite[11]等の配信性能の向上機能を組み合わせることが考えられる．しかし，このアプローチでは配信品質保証の実現が難しい．なぜならば，配信品質の保証のためには，配信に必要な資源の予約宣言等の既存OSのインタフェースでは提供されていない機能をアプリケーションが利用する必要があるためである．

一方，後者のアプローチとしては，splice[12]，RoadRunner[13]，Nemesis[14]，Tiger Video File Server[15]等が知られている．しかし，これらのアプローチは，開発工数の低減や市販ストリームサーバとの機能互換性維持について考慮していない．さらに，以下の点で本研究と差がある．

spliceとRoadRunnerは，デバイス間のデータ転送を高速に行う特別なパスを作成することにより，高性能なストリーム配信を実現する．しかし，このパスを経由してストリーム配信を行う場合，アプリケーションレベルのヘッダ（たとえば，RTPプロトコルを使用する場合はRTPヘッダ）を挿入したり，ディスクから読み込んだデータをネットワークに送信するタイミングを制御したりすることができない．HiTactix搭載の外付けI/Oエンジンでは，アプリケーションが自由に上記制御を行える．

Nemesisは，デバイスドライバ層にI/Oスケジューリング機能，すなわちI/O要求の実行順序を入れ換える機能を組み込むことにより，配信品質保証を実現する．しかし，I/O要求発行のタイミングの制御については考慮されていないため，I/O要求を発行するスレッドが厳密にスケジューリングされないことによる配信品質の劣化が発生しうる．HiTactix搭載の外付けI/Oエンジンは，「I/O実行スケジューリングテーブル」を保持することにより，この配信品質の劣化を防いでいる．

Tiger Video File Serverは，ディスクI/Oやネットワーク I/Oのスケジューリングテーブルを保持することにより，配信品質保証を実現する．しかし，ストリーム配信以外の負荷（たとえば，ストリームデータ登録にともなうディスク負荷）がかかった場合を想定しておらず，この場合に配信品質の劣化が発生しうる．HiTactix搭載の外付けI/Oエンジンは，3章で述べたとおり，このような場合にも配信品質を保証する．

7. まとめ

本稿では，既存の市販ストリームサーバの機能互換性を維持しつつ，当該サーバの配信性能の向上，配信品質保証機能追加を少ない開発工数で実現するストリームサーバの構成方式である外付けI/Oエンジン方式を提案した．本方式では，既存の市販ストリームサーバとHiTactixのような専用OSを搭載した外付けI/Oエンジンを連動させ，既存の市販ストリームサーバのストリーム配信処理のみを外付けI/Oエンジンに代行させる．

さらに本稿では，Darwinストリームサーバと HiTactix搭載の外付けI/Oエンジンを連動させた．この連動のために，HiTactixに，GDSLと呼ぶ低オーバヘッドでストリームデータの転送処理を行うライブラリを実装した．加えて，外付けI/Oエンジン方式の定量的な評価を行った．評価の結果，上記連動は9K行程度のコード追加で実現できること，また，連動オーバヘッドによるストリーム配信性能の劣化は1%以下に抑えられること，等を確認した．

参考文献

1) 阿蘇和人：ブロードバンドに命を吹き込め，日経コミュニケーション，No.343, pp.94–111 (2001).

2) nCube: n4 Streaming Media System, *nCube White Paper*, No.PN109786 (2000).
3) Venkatasubramainian, N. and Nahrstedt, K.: An Integrated Metric for Video QoS, *Proc. 5th ACM international conference on Multimedia*, No.208, pp.371–380 (1997).
4) 竹内　理ほか：連続メディア処理向き OS の周期駆動保証機構の設計と実装，情報処理学会論文誌，Vol.40, No.3, pp.1204–1215 (1998).
5) 中野隆裕ほか：Ethernet 上で QoS を保証する通信方法の設計と実装，情報処理学会論文誌，Vol.40, No.2, pp.322–332 (2000).
6) 竹内　理ほか：OS 接続モジュール Symbiose を用いた BSD-HiTactix 連動システムの設計と実装，情報処理学会研究報告，Vol.2000-OS-83, pp.31–36 (2000).
7) Iwasaki, M., Takeuchi, T., Nakano, T. and Nakahara, M.: Isochronous Scheduling and its Application to Traffic Control, *19th IEEE Real-Time System Symposium*, pp.14–25 (1998).
8) 岩嵜正明ほか：連続メディア処理向きマイクロカーネルの開発（1）～（5），第 53 回全国大会講演論文集（1），pp.141–150 (1996).
9) Schulzrinne, H.: Real Time Streaming Protocol (RTSP), RFC-2326 (1998).
10) Chu, H.K.J.: Zero-Copy TCP in Solaris, *Proc. USENIX Annual Techinical Conference*, pp.253–264 (1996).
11) Pai, V.S., Druschel, P. and Zwanepoel, W.: I/O-Lite: A Unified I/O Buffering and Caching System, *ACM Trans. Comput. Syst.*, Vol.18, No.1, pp.37–66 (2000).
12) Fall, K. and Pasquale, J.: Improving Continuous Media Playback Performance with In-Kernel Data Paths, *International Conference on Multimedia Computing and Systems*, pp.100–109 (1994).
13) Miller, F.W. and Tripathi, S.K.: An Integrated Input/Output System for Kernel Data Streaming, *SPIE/ACM Multimedia Computing and Networking*, pp.57–68 (1998).
14) Barham, P.R.: A Fresh Approach to File System Quality of Service, *7th International Workshop on Network and Operating System Support for Digital Audio and Video*, pp.119–128 (1997).
15) Bolosky, W.J., Fitzgerald, R.P. and Douceur, J.R.: Distributed Schedule Management in the Tiger Video Fileserver, *16th ACM Symposium on Operating Systems Principles*, pp.212–223 (1997).

(平成 14 年 9 月 30 日受付)
(平成 15 年 5 月 6 日採録)

竹内　理（正会員）

1969 年生．1992 年東京大学理学部情報科学科卒業．1994 年同大学大学院理学系研究科情報科学専攻修士課程修了．同年（株）日立製作所システム開発研究所入社．連続メディア処理向きマイクロカーネルの研究，特にリアルタイムスケジューリング方式，リアルタイム通信方式，異種 OS 共存技術，ストリーミングサービスアーキテクチャの研究に従事．

レ・モアル ダミエン

1972 年生．1995 年 ENSEEIHT（National Engineering School of Electrotechnics, Electronics, Informatics and Hydraulics of Toulouse, France）卒業．2000 年京都大学大学院情報学研究科通信情報システム専攻修士課程修了．同年（株）日立製作所システム開発研究所入社．連続メディア処理向きマイクロカーネルの研究，特にリアルタイムスケジューリング方式，サービス品質保証可能なシステムアーキテクチャ，ストリームサーバ向けミドルウェアの研究に従事．

坂東　忠秋（正会員）

1968 年東京大学工学部卒業，同年（株）日立製作所入社．日立研究所部長，中央研究所部長，システム開発研究所副所長を経て，現在システム開発研究所主管研究長．制御用計算機システム，画像処理，マイクロプロセッサ，並列型スパコン，マルチメディア，ブロードバンドコンテンツ配信等の研究開発に従事．1976 年スタンフォード大学 Electrical Engineering 修了．1984 年工学博士．1991 年～1993 年千葉大学非常勤講師．2002 年より電通大学客員教授．