2004－OS－95　(12)
2004／2／27

ストリーミングメディアのための優先度に基づくキャッシュ機構

内藤義郎†　高野了成†
品川高廣‡　吉澤康文‡

ストリーミングメディアのアクセスパターンは，シーケンシャル・等間隔などの特徴から比較的容易に予測可能である．本研究では，ストリーミングメディアに特化したシンプルかつ柔軟なキャッシュ機構の実現方式として，Priority-based Caching(PBC) を提案する．PBC では，優先度という概念を導入し，優先度の高いブロックをキャッシングするようにする．優先度は予測したアクセスパターンからストリーミングメディアの各ブロックに対して計算される．優先度での一元管理によりキャッシュ管理をシンプルにし，評価関数の変更によるキャッシュ方式の柔軟な変更を可能にする．本機構を Linux カーネルに実装し，仮想クライアントにより負荷実験を行ったところ，改変していない Linux に比べて，ディスク I/O を平均 60%削減し，ディスクボトルネックによる仮想クライアント数の上限を 91% 向上させた．

Priority-based Caching Scheme for Streaming Media

NAITO Yoshiro†, TAKANO Ryosei†
SHINAGAWA Takahiro‡ and Yoshizawa Yasufumi‡

The access pattern of streaming media can be predicted comparatively easily from the features, such as sequential and regular intervals. In this paper, Priority-based Caching (PBC) is proposed as a realization system of the simple and flexible cash mechanism which specialized in streaming media. In PBC, the concept of a priority is introduced and the block of high priority is cached. A priority is calculated from the predicted access pattern to each block of streaming media. Cache management is made simple by unified management with a priority, and flexible change of the cache system by change of an evaluation function is enabled. Load experiment between this technique and unmodified Linux shows that disk I/O was reduced an average of 60%, and the maximum of the number of virtual clients by the disk bottleneck was raised 91%.

1．はじめに

近年，家庭への PC の普及やネットワークの広帯域化，常時接続サービスの登場などに伴い，映像や音声などの連続メディアを配信するストリーミングサービスが普及しつつある．このため，より多くのクライアントに高品質のストリーミングサービスを提供するために，ストリーミングサーバはさらなる配信性能の向上が求められている．

†東京農工大学大学院工学研究科
Graduate School of Technology,Tokyo University of Agriculture and Technology
‡東京農工大学工学部
Faculty of Engineering, Tokyo University of Agriculture and Technology

ストリーミングサーバでは大容量のファイルを多数のクライアントへ配信する必要があり，そこで大量におこる入出力処理が配信性能のボトルネックとなりやすい．そこで，ディスク上のファイルをメモリ上にキャッシュすることで入出力を削減することが有効である．しかし，LRU(Least Recently Used)などの参照局所性を仮定するキャッシングアルゴリズムは，シーケンシャルアクセスに対してキャッシュヒット率が高くない．またこれらのホットスポットをだけを単なる LRU でキャッシングするアルゴリズムではストリームメディアファイルの構造，ビットレート，クライアントのアクセスパターンや負荷状況といったストリーミング特有の情報を利用して柔軟なキャッシュ管理をすることができない．

われわれの研究グループでは，上記のような従来のキャ

ッシュアルゴリズムが利用していなかったストリーミングメディアの特性を利用して，ボトルネックとなる I/O を削減して配信性能の向上に結びつけることを目標としたストリーミングサーバの研究を進めている．

本稿ではストリーミングメディアの新しいキャッシング方式として PBC(Priority-based Caching)を提案する．PBC では，優先度という概念を導入して，キャッシュポリシーとキャッシュメカニズムを分離する．これによりキャッシングメカニズムをシンプルにしつつ，キャッシュポリシーの柔軟な変更を実現している．

キャッシュポリシーは優先度ポリシーとして表されている．優先度ポリシーはキャッシュ単位であるブロックの優先度決定方針である．本稿で提案する優先度ポリシーはストリーミングメディアのアクセスパターンを予測して入出力の削減を行い，配信性能の向上に結びつける．

以下，2 章ではキャッシュ機構の設計方針を述べ，3 章で本稿で提案する PBC について述べる．4 章で PBC の Linux における実装法を示し，5 章で改変していない Linux と PBC との比較実験の結果を示す．6 章で関連研究に触れ，7 章で本稿をまとめる．

2．キャッシュ機構の設計方針

本章では，ストリーミングサーバのキャッシュ機構が満たすべき要件を述べ，本稿で述べる手法でその用件を満たすためのアプローチを述べる．本章では，キャッシュ機構の設計方針として，次の 3 つの点について述べる．

・ストリーミングシステムの抽象化
・優先度に基づくキャッシュ管理
・優先度ポリシー

2.1 ストリーミングシステムの抽象化

キャッシュ機構の設計に必要なストリーミングシステムの抽象化を行う．まず，キャッシュ管理の単位を"ブロック"と定義する．ブロックは固定長でディスク入出力の単位でもある．"クライアント" はストリームメディアの配信位置を抽象化したものである．一つのストリーミングメディアに対し複数のクライアントが存在し，一定の速度でストリームメディアの先頭から終端まで止まることなく進むものとする．ストリーミングシステムは複数のストリーミングメディアの配信を行う．各コンテンツ間には参照頻度の偏りがあり，人気のコンテンツにはクライアントが集中するものとする．ストリーミングメディアの容量は数百MB から数 GB ありキャッシュに収まらないものを想定する．したがってキャッシュ機構はメモリが足りなくなるとキャッシュ済みのブロックをスティールしてディスクから読み込んだ別のデータをキャッシングする必要がでてくる．このときにすぐに再利用の可能性のあるキャッシュをなるべくスティールしないようにするポリシーが必要である．

2.2 優先度に基づくキャッシュ管理

一般のストリーミングメディアには，シーケンシャルなアクセスが周期的に行われるため比較的容易にアクセスパターンが予測可能である．例えば，現在のクライアントの配信位置とビットレートから数秒後にストリームメディアのどの部分が必要になるかを予測できる．一般に一つのストリームメディアにおいて，すべてのクライアントへの配信位置が決まれば，数秒後のストリームメディアの全てのオフセットの参照可能性を予測できる．そこで，その参照可能性を基にストリームメディアのオフセットに優先度を設定し，優先度の高い部分をなるべくキャッシングするようにすればキャッシュヒット率を向上させられるはずである．

この方式にはいくつかの利点がある．まず一つ目は，キャッシングポリシーとキャッシングメカニズムを分離することが可能になることである．このためキャッシュポリシーを柔軟に変更できる設計が可能である．二つ目はキャッシュ管理がシンプルかつ効率的になることである．ストリーミングに特化したキャッシュ機構はストリーミングの特性を考慮した複雑な制御になりがちだが，本方式ではストリームメディアの特性を優先度として抽象化しているのでキャッシュ機構自体はブロックの優先度だけを判断するだけでよく，シンプルな設計にすることが可能である．

2.3 優先度ポリシー

優先度に基づくキャッシュ管理において，キャッシュヒット率の向上は優先度の設定で大きく変わるので優先度ポリシーの設計は重要である．ここでは優先度ポリシーの枠組みについて説明する．まず，優先度の算出は優先度関数によって行う．優先度関数はストリームメディアの情報を用いて，システムで用いる全てのブロックの優先度評価を行う．優先度関数は任意の時刻において，全てのストリームメディアのキャッシュの優先度を一貫して評価できる必要がある．優先度関数に影響のあるパラメータは各クライアントの位置，ストリームメディアのビットレートである．各パラメータの詳細は，3 章で説明する．

3．PBC：Priority-based Caching

本章では，2 章で述べた設計方針を基に設計した本稿で新しく提案するキャッシュ機構である PBC：Priority-based Caching について述べる．

3.1 PBC の構成

PBC では，図 1 に示すように，優先度の評価器と優先度

を決定するポリシーモジュールが分離する構造になっている．ポリシーモジュールはまず優先度関数を評価器に登録する．評価器は一定の周期でタイマーによって駆動され，優先度関数をコールバックしながらブロックプールにある全てのブロックの優先度を評価する．実際にキャッシングを行うのはキャッシュモジュールである．キャッシュモジュールはストリーミングメディアをキャッシュする必要がでたらブロックプールからブロックを割り当ててもらう．このときブロックプールは優先度の一番低いブロックを選択して，そのキャッシュを無効にしてからキャッシュモジュールに渡す．キャッシュモジュールは新たにディスクからブロックにデータを読み込んでキャッシュを有効にし，優先度を設定してからブロックプールにブロックを戻す．

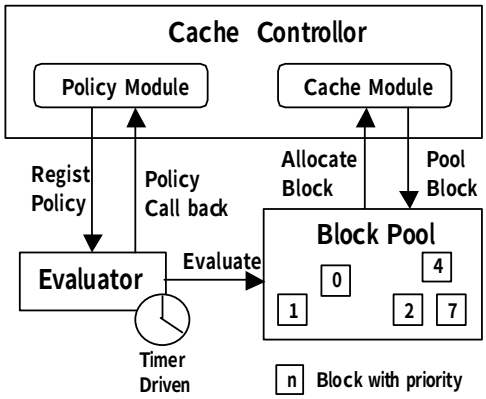

図 1　PBC モデル

3.2 優先度ポリシー

優先度の決定は優先度関数によって行う．ここでは優先度関数の設計に必要なポリシーを述べる．

クライアントの配信位置は優先度関数の形状を決める上で一番重要な情報である．クライアントはシーケンシャルで周期性のあるアクセスを行うため，配信位置より前方にあるブロックでは配信位置に近いほど早い時刻に参照される．もし配信位置に近いブロックをスティールしてしまえばディスク入出力が間に合わずストリーミングメディアのリアルタイム性を確保できなくなる．逆に配信位置から遠いブロックはスティールしてもディスク入出力でも間に合うのでリアルタイム性に問題はない．以上により配信位置より前方にあるブロックでは，配信位置に近いほど優先度を高くする．次に配信位置より後方にあるブロックについて述べる．配信位置より後方にあるブロックは配信が終わったブロックであり，将来の参照可能性が最も低い．よってこれらのブロックは優先度を一番低くする．

クライアントが複数いる場合は，前方のクライアントがディスクから読み込んだデータを後方のクライアントがそのまま再利用できればディスクに出す I/O の数が大幅に削減される．これを I/O ピギーバックと呼ぶ．全てのクライアントを I/O ピギーバックにできれば理想だが，キャッシュの大きさに限界がるので，I/O ピギーバックにするクライアントを選択する必要がある．従って I/O ピギーバックにするクライアントをいかに選択するかが I/O 削減による配信性能向上の重要なポイントである．そこで I/O ピギーバックにする選択基準としてメモリ効率という観点を重視した．つまり，クライアントを I/O ピギーバックにするためにピンダウンしなければいけないメモリ少ないほどいいという考え方である．いいかえると，クライアント間の距離が近いものを I/O ピギーバックにするということである．こうすることで制限されたメモリ量で最大数のクライアントを I/O ピギーバックにすることが可能になる．

以上により，本研究では図 2 のような優先度ポリシーを提案する．図 2 のような優先度ポリシーにすることで，クライアントの前方にあるブロックはクライアントに近いほどスティールされにくくなる．また後方にあるブロックは優先度が最低になるのですぐに再利用されるようになる．クライアントが近くなり優先度がぶつかった場合は大きいほうを選択する．こうすることで，クライアントが近づくにつれ，クライアント間のブロックは優先度が高くなり，メモリがピンダウンされやすくなる．従って，クライアント間が近いほど I/O ピギーバックにするというポリシーが表現できていることになる．

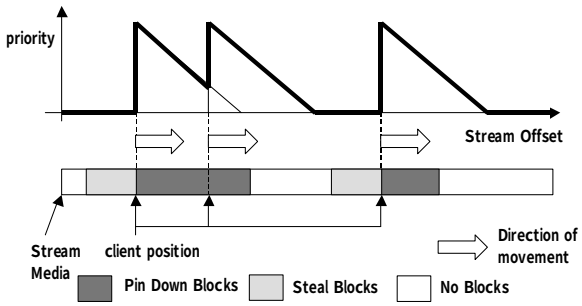

図 2　提案する優先度ポリシー

3.3 優先度関数

優先度関数を簡潔に述べると，あるストリームメディアの任意の位置 x を与えたときに，その位置に対応する優先度 P を返す関数である．優先度関数自体はストリームメディアのどの部分がキャッシングされているか評価段階まで分からないので，原理的にはストリームメディアの全ての位置において優先度が計算できるように設計する必要がある．本稿で提案するストリームメディア向けの優先度関数はクライアントの配信位置とストリームメディアのビットレートを基に形状を決定する数学的に定義された関数となっている．

3.3.1 優先度関数の定義

優先度関数の形状を決定するために，補助関数として，クライアント関数 C(x)を定義する．クライアント関数はストリームメディアに一つのクライアントしかいない場合の優先度関数である．C(x)の形状を決めるパラメータはクライアントの位置 p とビットレートにより決まる傾き定数 a である．傾き定数の詳細は 3.2.2 で述べる．このとき，ストリームメディアの位置を x とすると C(x)は次のようになる．

$$C(x)=\begin{cases} -a(x-p)+1 & (p \le x \le p+\frac{1}{a}) \\ 0 & (x < p,\ p+\frac{1}{a}) \end{cases} \quad \cdots\cdots①$$

この関数は，図 3 に示すように，優先度は 0 から 1 までの連続値をとり，p で一番高い優先度である 1 を返し，p+1/a まで線形的に減少する．また，このときの優先度が 0 でない区間を優先範囲呼ぶ．この関数を用いて複数のクライアントが存在するときの優先度関数 S(x)を定義する．クライアントの位置 p_1, p_2,・・・・・・，p_n が与えられたときのクライアント関数を C_1，C_2，・・・・・・,C_3 としたとき，S(x)は次のようになる．

$$S(x)=\max(C_i(x)) \quad (1 \le i \le n) \quad \cdots\cdots②$$

図 4 は②式を表したものである．図 4 の意味するところは，3.1 節で述べた優先度ポリシーと同じである．この関数によって，I/O ピギーバックの数を最大にして，高負荷時のディスクボトルネックの上限を大幅に引き上げることが可能となる．

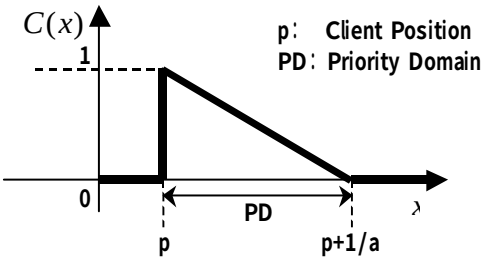

図 3 クライアント関数 C(x)

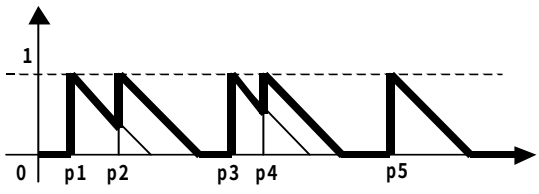

図 4 優先度関数 S(x)

3.3.2 ビットレートと優先度関数の形状

ビットレートが違うストリームメディアでは単位時間にクライアントが進む距離が違うので優先度の正規化が必要である．その役割を担うのが傾き定数 a である．傾きを求めるに当たって，クライアント関数の三角形の高さは同じなので傾きは優先範囲を決めれば決まる．ここで新たに優先期間 d というパラメータを導入する．このパラメータの単位は秒であり，ストリームメディアにおいてこの期間中にクライアントが進む距離を優先範囲と定める．これにより，任意のストリームメディアの傾き定数は，そのメディアのビットレートを b としたときに，次の式でもとめられる．

$$a=\frac{1}{db} \quad \cdots\cdots\cdots③$$

③式によりビットレートが高いと傾きが小さくなり，優先範囲が広くなる．ビットレートと傾きの関係を図 5 に示す．

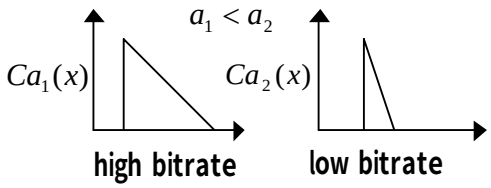

図 5 傾き定数とビットレート

4．PBC の実装

本章では PBC を実現する手法として，Linux カーネル内への実装法を示す．

4.1 実装の概要

システム構成は図 6 に示すように，Session，Client，Stream， Block Pool，Priority-Eval，Read-Ahead，Proc の 7 つのモジュールから構成される．

Stream はストリームメディアごとに存在し，対応するストリームメディアのキャッシュを管理する．このモジュールは PBC の主要的な機能を果たし，図 1 のポリシーモジュールとキャッシュモジュールに相当する．Client はクライアントの配信位置や読み出しバイト数などを保持する．Session は Stream と Client の生成と破棄を”セッション”という単位で管理するためのモジュールである．Block Pool と Priority-Eval は PBC の同名の機能を実現するためのモジュールである．Read-Ahead はブロックのディスクを有効に活用するための先読みを実現するものである．

Proc は PBC の各種統計情報を proc ファイルシステムを通して表示可能にするモジュールである．以下，PBC を利用するためのプログラミングインタフェースと各モジュールの実装について述べる．

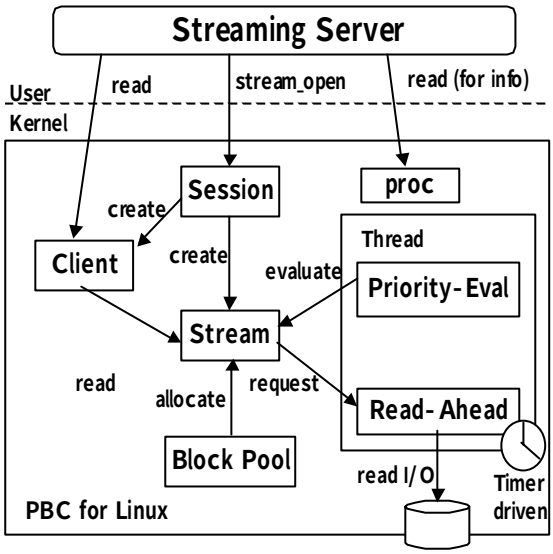

図 6　システム構成

4.2 プログラミングインタフェース

PBC ではクライアントとストリームメディアを対応付けるために，次に示すような **stream_open** というシステムコールを追加する．

表 1　PBC を利用するためのシステムコール

説明	システムコール
セッションを開始する	int stream_open(char *filename , long long offset, long long length, int bitrate)

このシステムコールは表 1 に示すように，引数にストリーミングメディアのファイル名と，ストリーム範囲を指定するためのオフセットとサイズ，そしてストリームメディアのビットレートを指定する．関数の戻り値はストリーム記述子である．このストリーム記述子に対して，通常の Unix プログラミングと同じように read システムコールを行うことでカーネル内の PBC を用いたストリームメディアの読みこみを行うことができる．また，読み出し位置の移動と stream のクローズにも通常の Unix プログラミングと同じ lseek と close システムコールを用いることができる．図 7 に簡略化したストリーミングサーバを例にシステムコールの使用法を示す．

```
vsd = stream_open(filename, video_start,
                  video_size, video_bitrate);

while(!eof_streaming) {
    /* transfer RTP packet*/
    lseek(vsd, rtp_payload_pos, SEEKSET);
    read(vsd, rtp_payload, rtp_payload_size);
    vec[0].iov_base  =  rtp_header;
    vec[0].iov_base  =  rtp_payload;
    writev(vsock, vec, 2);
}
```

図 7 プログラミング例

4.3 キャッシュ管理の実現

キャッシュ管理は Stream モジュールで実現している．Stream モジュールでは対応するストリームメディアのキャッシュを格納したブロックをリストとしてもっている．このブロックリストは格納しているキャッシュのファイルオフセットで整列されている．read アクセスがあった場合は現在のクライアントの位置に対応するブロックを起点として，要求されたオフセット部分を格納しているブロックを線形に探索する．シーケンシャルアクセスの場合，先読みが成功していれば探索は 1 回で終了する．

4.4 ブロック管理の実現

ブロック管理を実現しているのは Block Pool である．Block Pool では本機構の初期化時にキャッシュに使う全ての物理ページフレームを確保してページプールを作成する．さらにそのページフレームいくつかまとめてブロックにマッピングする．このマッピングは初期化時に一度だけ行われる．本機構ではブロックサイズはページサイズ単位で変更することが可能である．物理ページフレームはディスクからのキャッシュの格納時やクライアントのキャッシュ読み出し時に必要に応じて仮想アドレスにマッピングする．

Block Pool は，優先度付けされたブロックを管理するために，優先度レベル別リストを持っている．優先度レベルはデフォルトで 8 レベルになっている．優先度レベル別リストの各レベルには対応する優先度を持つブロックが繋がっている．ブロックプールは，キャッシュ飽和時にブロックを要求された場合，レベル 0 から順にレベルをあげていき，最初にブロックが見つかったレベルからブロックスティールを行う．

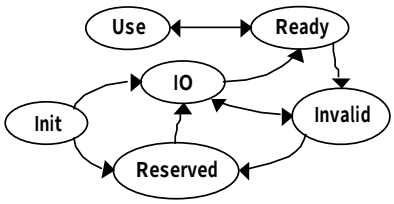

図 8 ブロックの状態遷移

ブロックは図 8 のようないくつかの状態を持っている．まず最初にブロックにキャッシュがない場合がInitである．ブロックはいったんキャッシュされると二度とInit状態になることはない．つまりこの状態のブロックが存在するのはシステム稼動時の最初の期間だけで，キャッシュが飽和した後は Init 状態のブロックは存在しない．Reserved は先読みリクエストを発行されたブロックである. IO は実際にディスク入出力を行っている状態でディスク入出力が終るとキャッシュが利用可能な状態である Ready に変わる．このブロックをクライアントが使用するときには USE 状態に変え，使用が終るとまたReady状態に戻す．キャッシュが飽和した状態でスティールが行えるのは Ready 状態のブロックだけである．ブロックがスティールされると一瞬 Invalid 状態に変わり，新しいキャッシュをディスクから読み込むために，IO またはReserved状態に変わる．

4.5 優先度評価の実現

優先度評価のためにはクライアント関数を実現する必要がある．クライアント関数は図9に示す優先度テーブルとして実現されている．優先度テーブルはストリーム毎に一つ存在し，ビットレートと優先度期間から優先範囲が計算され，その範囲内に0でない値が設定されている．クライアントが現在いる位置が最高優先度(図 9 では 7)になり，右にいくほど低くなる．ブロックの優先度はもしブロックがどれかのクライアントの優先範囲に属していれば，対応する優先度テーブルの値がそのブロックの優先度になる．もし優先範囲が重なっている場合は高いほうの優先度を選択する．優先範囲に属さないブロックは優先度0になる．

優先度関数はストリーム作成時に優先度評価モジュールに登録される．登録された関数は専用のカーネルスレッドにより，一定周期(デフォルトで 1/4 秒)間隔でコールバックされ，ブロックプール内のブロックを再評価する．優先度関数は存在する全てのストリームをたどって，そのストリームに所属するブロックの優先度を再評価する．ストリームに所属するブロックの優先度評価法は，上でも述べた通り，クライアントの優先範囲内にあるブロックを優先度テーブルから引いて評価する．ストリームに属する全てのクライアントについて評価が終ったあとで，もしブロックの優先度が変わっていたら，適切な優先度別リストに繋ぎ変える．

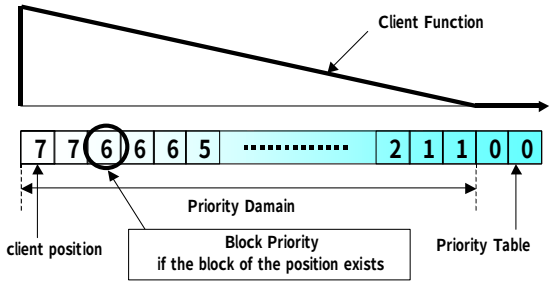

図 9 優先度テーブル

4.6 統計・管理情報の表示

統計･管理情報はprocファイルシステムを利用して確認できるようにした. その構成は図10のようになっている．streams ディレクトリには現在アクティブなストリームのディレクトリがありその中に実際の管理情報である stream_prof ファイルとそのストリームを読んでいるクライアントのファイルがある．クライアントのファイルは clients ディレクトリにシンボリックリンクが張られている．init は本機構の初期化パラメータを読み書きするファイルである, ctl ファイルは本機構の起動と終了を制御する．このファイルに”1”を書き込むことで本機構が起動し，”0”を書き込むことで終了する．block_pool と pbc ファイルは管理情報と統計情報を表示する．

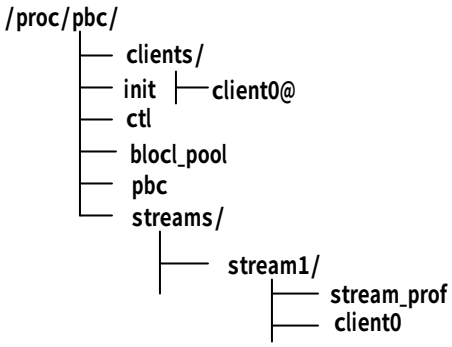

図 10 proc を使った情報表示

5．性能評価実験

PBC を適用した場合と改変していないLinuxとの間で，リードレート及びI/Oレートの比較を行った．実験環境を表2に，使用したハードディスクの詳細を表3に示す．以

下，評価方法と評価結果について述べる．

表 2　実験環境

Machine	PC/AT 互換機
CPU	Athlon MP 1800+
Memory	2GB
HDD	120GB
OS	Linux Kernel 2.4.21

表 3　使用ハードディスクの詳細

容量	120GB
回転速度	7,200rpm
バッファサイズ	2MB
外部転送速度(MB/sec)	ATA/133
平均シークタイム(ms)	<8.8
平均回転待ち時間(ms)	4.17

5.1 評価方法

性能評価は，仮想クライアントによって負荷をかけることで行う．仮想クライアントはメディアのビットレートにしたがって read を発行する．1 回の read サイズは 256KB 固定とし，それをバッファリングする．実験はネットワークを介さずにサーバとクライアントを同一マシン上で動作させることで，ネットワークによる影響を排除する形で行った．

負荷は表 4 に示すストリーミングメディア 10 本を仮想クライアントが読み出すことによってかける．各タイトルの参照頻度を図 11 に示す．この参照頻度は人気集中型アクセスを表すモデルとして比較的よく用いられる Zipf 分布をもとに作成した．

キャッシュサイズは，改変していない Linux と PBC を適用した Linux とで同じく 1.4GB とした．また，PBC のブロックサイズは 128KB とした．

表 4　ストリーミングメディア

フォーマット	MPEG4 シンプルプロファイル
再生時間	116 分
映像サイズ	320×240
ビットレート	138KB/sec (固定)
ファイルサイズ	940MB

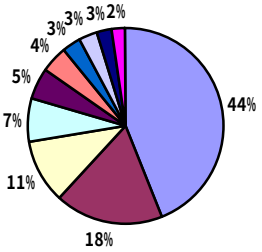

図 11 各タイトルの参照頻度

5.2 結果と考察

仮想クライアント数を平均到着間隔 6 秒のポアソン分布で増加させた場合のクライアント数とシステム全体のリードレート(MB/sec)を図 12 に示す．また，そのときの時間軸上における I/O レート(回/分)を図 13 に示す．

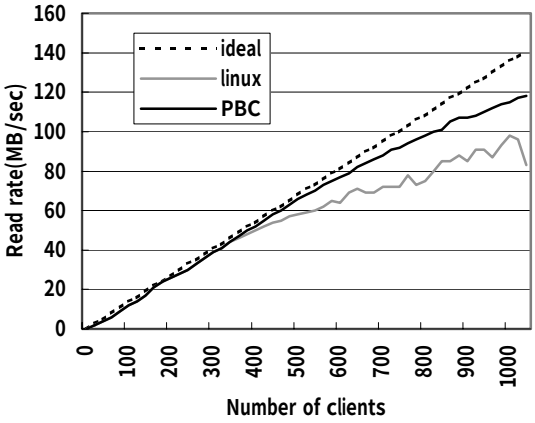

図 12　クライアント数とリードレート

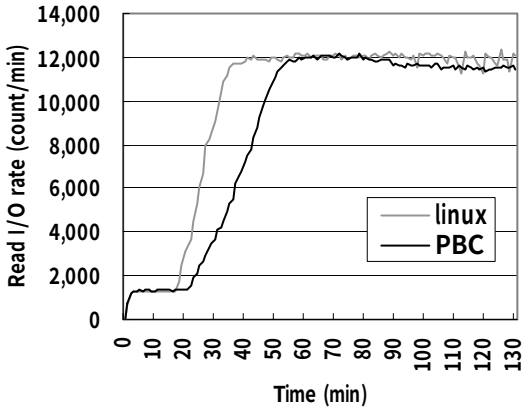

図 13　I/O レートの時間変化

図 12 が示すように，PBC の方が改変していない Linux と比べてリードレートの劣化が少ない．ストリーミングの場合リアルタイム性が重要となるのでリードレートの劣化がどこまでも許容されるわけでわないので，今回は理想の

リードレートよりも 10%低下したときのクライアント数を最大サポートクライアント数と定義する. このとき図 12 より，PBC ではリードレートが 10%低下したときのクライアント数が 880 であるのに対し，改変していない Linux では 460 である．したがって，PBC は Linux 標準よりも最大サポートクライアント数が 91%向上したと言える．

次に I/O レートの検証を行う．図 13 から，PBC, Linux ともに最初の 3 分間くらいで I/O レートが急増しその後，20 分前まで両者とも I/O レートが一定になっているのがわかる．これは最初 10 本のタイトルにおける先行クライアントのために，I/O レートが急増し，その後，後続クライアントはメモリスティールが起きていないので I/O ピギーバックする形になり，I/O レートが一定になっていることを示している．その後 Linux では 17 分，PBC では 22 分ごろから I/O レートが急増する．これはキャッシュを使い果たし，メモリスティールが起きるようになったので，一部の後続クライアントが I/O ピギーバックできなくなり，ディスク I/O が急増しているものと思われる．PBC では，linux 標準より I/O レートが急増する時間が 5 分遅いのでそのぶんキャッシュが有効に働いていることを示している. I/O レートの伸びに関しては Linux と PBC では差が顕著に表れている．時刻が 30 分のときの両者の I/O レートを比較してみると，PBC は 3676 回/分であり，Linux 標準の場合は，9487 回/分である．このことから PBC は Linux 標準よりディスク I/O を 61%削減していることがわかる．

以上のことから, PBC の I/O ピギーバックを有効活用する優先度ポリシーがディスク I/O 削減し，ディスクボトルネックに達する時間を遅らせたといえる．またこの結果，前述の最大クライアント数の増加につながったと言える．

6．関連研究

本研究で提案した優先度ポリシーは，クライアント同士が近い場合に後続クライアントを I/O ピギーバックにするという点で，Interval Caching Policy (ICP)[1),2),3)]やスパニンググループキャッシング(SGC)[5)]のアプローチに近い. ICP は同一コンテンツに対する二つのクライアントの要求間隔が短い場合，先行クライアントがディスクから読み込んだデータを後続クライアントのために，ディスクに残しておくというものである．ICP ではキャッシュ単位がインターバル単位なのに対し, PBC ではそれよりも細粒度のブロックという単位を採用しているので ICP よりも，メモリスティール時のペナルティも少なくクライアント負荷の変動に強い．SGC は，われわれの研究グループが提案したキャッシング機構である．この手法ではキャッシュ単位は固定長のブロックであり，クライアント間のブロックをスパニンググループという単位で管理している．そして，クライアント間の距離が長いスパニンググループからブロックを解放する. SGC もブロックを単位としたメモリ効率重視の I/O ピギーバックを特徴としているが，本論文では，その発展として，キャッシュポリシーの柔軟な変更が可能であり，優先度の一元管理により，シンプルなキャッシュ機構を実現している．Damien Le Moal らは，ストリームサーバ専用 OS である HiTactix を外付け I/O エンジンとして利用して，Linux 上のストリーミングサーバと協調動作することで，高速入出力による配信性能の向上を実現している [4)]．これに対し，本研究では入出力削減を目的としたキャッシュ機構を実現することで配信性能の向上を図っている．

7．おわりに

本論文では，ストリーミングメディアに特化したキャッシュ機構の実現方式として, Priority-based Caching(PBC) を提案した．PBC では，優先度という概念を導入し，優先度ポリシーとキャッシング機構を分離することでシンプルかつ柔軟な機構を実現した．また，I/O ピギーバックを有効活用して I/O の削減を行う優先度ポリシーも提案した. 上記の提案手法の実現法として Linux カーネルへの実装法について述べた. 実験により改変していない Linux 比べて，ディスク I/O を平均 60%削減し，ディスクボトルネックによる仮想クライアント数の上限を 91%向上させたことがわかった．このことから PBC のメカニズムと優先度ポリシーの効果が実証できた.

参考文献

1)A. Dan, D. Sitaram. Buffer Management Policy for an On-Demand Video Server. IBM Research Division, RC 19347,Jan1993

2)A. Dan, Y. Heights, and D. Sitaram. Generalized interval caching policy for mixed interactive and long video workloads. In Proc. of SPIE/A CM Conf. on Multimedia Computing and Networking, pp. 344-351, May 1996.

3)A.Dan and D.Sitaram. Multimedia caching strategies for heterogeneous application and server environments. Multimedia Tools and Applications, vol. 4, pp. 279-312, May 1997.

4)Damien Le Moal,Tadashi Takeuchi,Tadaaki Bandoh. Cost-Effective Streaming Server Implementation Using Hi-tactix. in Proc the tenth ACM international conference on Multimedia,pp382-391, December 2002.

5)高野, 浅見, 帆波, 吉澤.ストリーミングデータの参照特性に基づく入出力削減方式の提案.情報処理学会研究報告 2003-OS-92 Vol.2003,No.19 pp.61-68 Feb 2003.