クラスタ型分散ファイルシステムに対応した協調型キャッシュアルゴリズム

数藤義明†

本論文は，クラスタ型の分散ファイルシステム内においてファイルブロックのキャッシュを有効に活用するための協調型キャッシュアルゴリズムについて述べたものである．クラスタ型の分散ファイルシステムは，複数のマシンとそれに接続されたディスクを利用して，非常に高い処理性能と耐故障性能を両立して実現することを目標としている．この分散ファイルシステムの処理性能を高めるために，クラスタ内の各マシン上に存在するキャッシュを有効に活用し，ネットワークでファイルブロックを転送するのに比較して非常に低速なディスクからの読み込みを低減するための協調型キャッシュアルゴリズムが望まれている．本論文では，クラスタ型分散ファイルシステム内のキャッシュ一貫性管理機構に組み込まれた新しい協調型キャッシュアルゴリズムを提案し，その評価を行った．その結果，従来の協調型キャッシュアルゴリズムよりも，ディスクアクセス回数を最大で60%低減できることを確認した．

A Cooperative Caching Algorithm for Cluster Distributed File Servers

YOSHIAKI SUDO†

This paper describes a cooperative caching algorithm. This algorithm aims to raise performance of cluster distributed file servers using idle server machines' memory as a cooperative file block cache. The algorithm is specially designed for cluster distributed file servers. It is integrated in the cache coherence protocol that should be included in cluster distributed file servers. Moreover, in order to realize efficient use of these file block caches on all server machines in a cluster system, this algorithm aggressively discards non-master copies of file blocks in its cache memory using a new replacement algorithm called LRU with priority. In this paper, we explain the design and implementation of our new cooperative caching algorithm. We also present the performance evaluation result. The result shows that the algorithm is efficient for improvement of total hit ratio in the distributed file system and reduction of the number of disk access.

1. はじめに

従来のクライアント/サーバ型の分散ファイルシステムにおいては，クライアントからのファイルアクセスの要求はいくつかのサーバマシンに集中する．そのためサーバマシンのパフォーマンスが低いと，分散ファイルシステム全体のパフォーマンスを低下させてしまうという大きな問題点がある．多くのクライアントマシンを抱え，ファイルサーバへの多くのアクセス要求に対応する必要のある場合，高速なディスクやRAID，多くのメインメモリを備えた高性能で高価なサーバマシンが必要となっていた．さらに1台のマシンではアクセス要求に耐えられない場合には，複数のサーバマシンを用意し負荷を分散させる必要があった．

このような従来のクライアント/サーバ型の分散ファイルシステムの欠点に対して，クラスタ型の分散ファイルシステムが提案されている．クラスタ型の分散ファイルシステムでは，複数のマシンがファイルサーバとして動作し，かつそのサーバクラスタは従来の単一のファイルサーバと同様に扱える．またクライアントとサーバという区別をまったくなくし，システム内のすべてのマシンが分散ファイルシステムの1つの要素となるサーバレス型の分散ファイルシステムにも発展できる．以上のようにクラスタ型の分散ファイルシステムでは，システム内のマシンの持つディスクをすべてのマシンで共有し，耐故障性やスケーラビリティの高い分散ファイルシステムを構築可能なものとして

† キヤノン株式会社情報メディア研究所
Media Technology Laboratory, Canon Inc.

いる．

さらに，クライアント/サーバ型の分散ファイルシステムにおいて，協調型キャッシングと呼ばれるキャッシングアルゴリズムが提案されている．これは，ファイルアクセスの多いクライアントマシンのファイルキャッシュから溢れたファイル自体，もしくはそのファイルの一部のファイルブロック☆を，アクセス要求の少ないクライアントマシンにネットワークを用いて転送し，元のクライアントマシンや他のクライアントマシンがそのファイルを再利用する場合に，サーバマシンから再び読み込むのではなく，転送したクライアントマシンのキャッシュから読み込んで利用するキャッシング方式である．クライアント間でのファイルブロックの転送を利用することで，すべてのクライアントマシンのキャッシュを合わせて活用することで非常に大容量のファイルキャッシュを利用することが可能となる．文献2)に示されているように，クライアントマシン間の広帯域ネットワークによるブロックの転送よりもディスクアクセスは非常に低速である．したがって，この協調型キャッシングを利用し，クライアントマシンからサーバマシンへのアクセス要求を低減して，サーバマシンによるディスクアクセス回数を低減することによって，分散ファイルシステムの処理能力を高めることが可能となる．

本論文は，クラスタ型の分散ファイルシステムの1つであるShared Logical Disk[8]（SLD）のキャッシュ一貫性管理機構に組み込んだ新しい協調型キャッシュアルゴリズムである優先度付きLRUアルゴリズムを提案し，どの程度サーバマシンのディスクアクセス回数を低減できるかの評価を行った結果を述べる．以下，2章では，従来のクラスタ型の分散ファイルシステム，および従来の協調型キャッシュアルゴリズムについて触れ，3章では，本論文のターゲットシステムであるSLDシステムの概要を説明する．4章では，提案する協調型キャッシュアルゴリズムについてを述べ，5章ではその評価を行う．最後に6章において，本論文のまとめを記す．

2. 従来の研究

2.1 クラスタ型分散ファイルシステム

クラスタ型の分散ファイルシステムとしては，単にRAIDをSCSIバスによって複数のマシンで共有するものから，Zebra[4]やPetal[5]のように，広帯域ネットワークで接続された複数のサーバマシンのディスクを利用して，分散ストライピングで高性能化や，ミラーリングなどの手法で冗長性を持たせたりしたものがある．UCBのZebraでは，クライアントマシン側でログ構造化ファイルシステム[6]におけるログファイルを分割し，複数のサーバにそれらを各々転送しディスク書き込みを行うことで高性能化している．また，DEC社のPetalは，サーバマシン間でファイルクラスタ単位でストライピングし，さらに2つのサーバ上にミラーリングすることによって冗長性を持たせているのが特徴である．

またクライアント/サーバをまったく区別しないサーバレス型の分散ファイルシステムとして，UCBのxFS[1),10)]やDEC社のFrangipani[9)]がある．xFSは，オリジナルのサーバレス型の分散ファイルシステムであり，システム内の各マシン上に分散したストレージサーバとマネージャ，クライアントなどの各エンティティによって構築されている．これらのエンティティが，システムの負荷や資源の配置状況などに従って分散配置されている．また，Frangipanniは上記のPetalを利用したxFSに類似したサーバレス型の分散ファイルシステムである．どちらも，システムに複数のマネージャが存在し，ファイルの配置情報の管理や，キャッシュの一貫性管理などを分散して行う．

これらはすべて，複数のファイルサーバを用意して高い可用性を実現し，ファイルの記録に冗長性を持たせることで高い耐故障性能を実現する．また，ファイルを分割して複数のファイルサーバで読み書きを行うストライピングを行い，さらにログ構造化ファイルシステムによる高い性能を引き出しているものもある．

2.2 協調型キャッシング

協調型キャッシングは，クライアント/サーバ型のファイルシステムにおいて，複数のクライアントマシン上のキャッシュを有効利用し，ファイルブロックをクライアントマシン間で転送することにより，サーバマシン上のディスクアクセスを低減し，分散ファイルシステムの処理能力を改善することを目標としている．システム内のどのクライアントマシンのキャッシュ上にも，あるブロックがキャッシングされていない場合には，サーバマシン上のディスクからそのファイルブロックを読み込んでくる必要がある．しかし，高速なサーバマシン上のディスクからの読み込みでも，ネットワークで転送するのに比較して数十倍の読み込み時間が必要である．したがって，一度ディスクから読み込んだファイルブロックは，廃棄するよりも，他のク

☆ ファイル自体を転送するか，一部のファイルブロックを転送するかは，実際のキャッシングの単位により異なる．本論文では数KBのファイルブロックをキャッシングの単位と仮定し，転送の単位もそのファイルブロックとする．

ライアントマシン上のキャッシュに転送しておき，そこから読み込むことができれば，再びディスクから読み込むよりもオーバヘッドが少なくてすむ．そのためには，クライアントマシン上のキャッシュから置換され廃棄されるファイルブロックが，他のクライアントマシンにキャッシングされているか，もしされていないならばそのブロックを他のクライアントに転送する必要があるかどうかなどを判別する機構が必要となる．また転送する場合には，どのクライアントマシンへ転送するかなどを決定する方針が協調型キャッシングアルゴリズムとして重要な要素となる．

N-chance forwarding アルゴリズム[2]は初期に提案された簡単な協調型キャッシュアルゴリズムである．このシステムでは，クライアントマシンのメモリにはブロック単位でファイルがキャッシュされている．全クライアントのキャッシュ内にブロックのコピーがないファイルブロックを *singlet* と呼ぶ．*singlet* をキャッシュ内から置換する場合には，ランダムに選択された他のクライアントマシンに転送し，そのファイルブロックがクライアントマシンのキャッシュ内から削除されないようにする．ただし，*singlet* を N 回以上転送した場合には，それ以上転送せずに廃棄する．これによってクライアントマシンのキャッシュが *singlet* によって埋まってしまうことを防止する．

GMS[3]（Global Memory Service）では，分散ファイルシステムではなく，分散メモリ管理システム内の協調キャッシュアルゴリズムを提案している．このアルゴリズムも，N-chance forwarding アルゴリズムの *singlet* と同様な状態を持ち，それに従ってキャッシュブロックを他のクライアントマシンに転送するか，廃棄するかを決定する．しかし，GMS では N-chance forwarding アルゴリズムがランダムに転送先を決定していたのとは異なり，システム内の各キャッシュバッファの最後にアクセスされた時刻を管理しておき，それに基づいてグローバルな LRU アルゴリズムを用いて転送先を決定する．

ヒントに基づくアルゴリズム[7]は，上記の GMS のグローバルな LRU アルゴリズムとは異なり，分散したマシン間で各マシン上のキャッシュブロックに関する情報をヒントという形で管理し利用するアルゴリズムである．上記の GMS や N-chance forwarding アルゴリズムでは，各キャッシュブロックのコピーがどこにあるかなどの情報を管理マシンが完全に把握していたが，このアルゴリズムでは管理マシンはキャッシュブロックの現在の厳密な情報を管理せず，古い情報をヒントとして利用する．キャッシュブロックに関する厳密な情報管理によって，非常に多くのメッセージと大きなオーバヘッドが必要になるが，このアルゴリズムではこれらが大きく低減される．また置換アルゴリズムも，キャッシュブロックのアクセス情報が GMS のように集中管理されてはおらず，各マシン間で転送された情報を用いてグローバルな LRU アルゴリズムの近似を行う．情報を分散管理し LRU アルゴリズムを分散して実行するため，厳密な LRU アルゴリズムは実現できないが，十分に近似は可能である．

これらのアルゴリズムは，すべてクライアント/サーバ型の分散ファイルシステム（GMS では分散メモリ管理システム）におけるクライアント間の協調型キャッシュアルゴリズムである．しかし，クラスタ型の分散ファイルシステムの協調キャッシングアルゴリズムに関しても，*siglet* と同じ概念を用いたファイルブロックのオリジナルコピーの管理方式や，転送するクライアントマシンの選択方法などで同様なアルゴリズムが適用可能である．

3. Shared Logical Disk の概要

Shared Logical Disk（SLD）はプリンストン大学の Shrimp プロジェクトにて開発がすすめられている分散ファイルシステムである．**図 1** に示されるよう

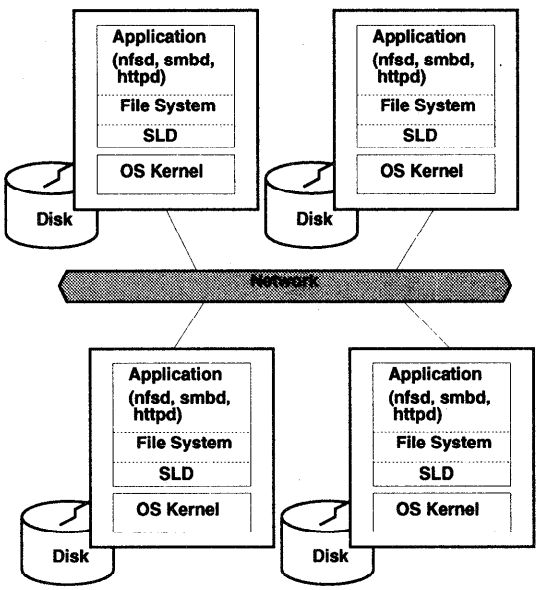


図 1 Shared Logical Disk の論理構成：分散システム上で仮想的なディスクを実現する論理ディスク層と，その上でファイルシステムを実現する層の 2 階層に分離されている．

Fig. 1 Shared Logical Disk logical structure: The lower layer is a virtual disk layer that controls disks, cache blocks, and its consistency with network. The upper layer is a file system layer that realizes user file operation such like open, read, write, and so on.

に，SLDはユーザアプリケーション内で動作するライブラリとして実装されており，大きく論理ディスク層とファイルシステム層が分離されている．論理ディスク層は互いに通信し合うことで，分散したマシン上の複数のディスクを，あたかも論理的な1つの大きなディスクとして見せ，ファイルシステム層に提供する．ファイルシステム層では，論理ディスク層によって提供された論理ディスクにアクセスするインタフェース関数を用いて，ファイルシステムを構築する．SLDのアプリケーションとしては，NFSサーバやSMBサーバ，さらにWWWサーバが実装されている．

3.1 論理ディスク層におけるファイルブロック間の依存管理

SLDの論理ディスク層（以下単にSLDとする）では，ファイルシステム層に対して論理ディスクブロックのリード/ライトの基本機能だけでなく，ファイルブロックのロック機構やファイルブロック間の依存関係の指定機能を持つ．ファイルブロックのロック機構は，複数のマシン上で動作するファイルシステム層が同じファイルブロックの書き換えを安全に行うために必要となる．

また，SLDではファイルブロックが複数のマシンのディスク上に配置されているため，通常は書き込みを行った2つのファイルブロックが実際にディスクに書き込まれる順序は保証されない．もし，すべての書き込みに関してこの順序保証を行うと，書き換えられたすべてのファイルブロックのディスクへの書き込みが直列化してしまい，非常に大きなオーバヘッドとなる．一方，書き換えられた複数のファイルブロックの書き込みに対して順序保証をしないとなると，ディスクやマシンの故障の際，先に書き込んだはずのブロックが書き込まれてない場合には，ファイルシステムの一貫性が保持できずファイルシステムの復旧が困難になる．そのために，SLDではファイルブロック間の依存関係を指定するインタフェースが用意されており，これを利用してファイルシステム層はファイルブロック間の依存関係を指定し，実際にディスクに書き込む際の順序を制御する．通常は，これを用いてファイルシステムのメタデータとファイル本体の依存関係の指定を行う．

3.2 ブロックキャッシュの一貫性管理

SLDのブロックキャッシュの一貫性管理は，図2に示されるように一般的に分散ファイルシステムで用いられるトークン方式を利用している．各ファイルブロックには，そのブロックのキャッシュ管理などを行うマシン（オーナマシン）が固定的に決められており，ブロックの内容はそのオーナマシンに接続されたディスクに記録されている．オーナマシンは，他のマシンからのブロックへのアクセス要求を受付けた場合，そのアクセスがリードアクセスかライトアクセスかに応じて，それぞれリードトークン，ライトトークンをファイルブロックとともに渡す．ただし，そのマシンがすでにファイルブロックを保持している場合には，トークンだけが渡される．

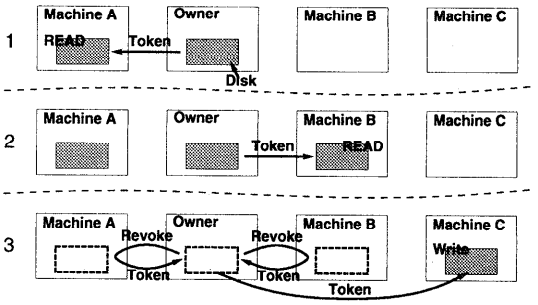


図2 SLDのトークンによるキャッシュ管理：1. マシンAがリードトークンを得る．2. マシンBがリードトークンを得る．3. マシンCのライト要求により，マシンAとマシンBのリードトークンが回収され，マシンCにライトトークンが渡される．

Fig. 2 SLD uses read/write tokens to manage cache consistency. 1. The machine A receives a READ token from the onwer machine. 2. Then, the machine B receives a READ token, too. 3. The machine C sends a write request to the owner machine. The owner machine revokes READ tokens from the machine A and B, and then sends a WRITE token to the machine C.

トークンを渡されたマシンは，そのトークンの種類に従ってアクセスを行い，利用後もそのトークンを保持してその後のアクセスに備える．オーナマシンは，複数のマシンからのリード要求に対してはリードトークンをすべてのマシンに渡す．しかし，ライト要求に対しては，ブロックのキャッシュの内容の一貫性を保持するために，すでにリードトークンを渡したマシンに回収要求を転送し，すべて回収し終わった後にライト要求のあったマシンにライトトークンを渡す．さらに複数のライト要求があった場合には，そのライト要求を直列化して処理する．

オーナマシンは複数のマシンから同じブロックへのリード要求がきた場合に備えて，一度リード要求の来たブロックは自分のキャッシュ内に保持しておく．もし自分のキャッシュ内に要求のあったブロックが存在せず，他のマシンのキャッシュに存在する場合には，ブロックの回収（リードトークンは回収しない）を行い，要求を送ってきたマシンにブロックを渡す．また，自分のキャッシュ内に要求のあったブロックが存在せず，さらに他のマシンのキャッシュにも存在しない場合に

はディスクから読み込みを行い，要求を送ってきたマシンにブロックを渡す．

4. 協調型キャッシュアルゴリズム

4.1 マスタートークンによるブロック管理

前章で述べたSLDのキャッシュの一貫性管理機構では，あるブロックのオーナマシンは，どのマシン上にそのブロックがキャッシングされているかなどの情報を厳密に管理している．しかし，各マシン上でローカルに置換アルゴリズムに従って選択されたキャッシュブロックは，トークンがオーナマシンに転送された後に即座に削除されてしまう．協調型のキャッシュを実装するためには，システム内で唯一のコピーであるキャッシュブロックが置換対象に選択された場合に，それを削除せず他のマシンに転送する必要がある．このため，従来の協調型キャッシュアルゴリズムと同様にキャッシュブロックがシステム内で唯一のコピーであるかどうかを，各マシン上で判別する機構，もしくはオーナマシンに問い合わせる必要がある．SLDでは，ヒントに基づくアルゴリズムのマスターコピーの概念と同様なマスタートークンを導入した．

マスタートークンは，あるマシン上のキャッシュ内にあるファイルブロックを削除せずに保持しておくべきこと示すトークンである．あるファイルブロックが置換対象になった場合に，そのファイルブロックがマスタートークンを持っている場合（そのブロックをマスターブロックと呼ぶ）には，そのファイルブロックが持っている他のトークン（リードトークンまたはライトトークン）とともにマスタートークンとファイルブロック自身をオーナマシンに返送する．逆にそのファイルブロックがマスタートークンを持っていない場合（そのブロックをノンマスターブロックと呼ぶ）には，持っているトークンをオーナマシンに返送したあとに削除してよい．また，オーナマシン上でマスターブロックが置換対象になった場合には，グローバルな置換アルゴリズムに従って，削除されるか他のマシンに転送されるかが決定される．他のマシンに転送されたファイルブロックが，そのマシン上の置換アルゴリズムに従って置換対象となった場合には，再びオーナマシンに返送される．

この方式においては，マスタートークンも他のトークンと同様にオーナマシンが管理し，マスタートークンは必ずオーナマシンを経由して他のマシンに転送されることになる．このように，マスターブロックは必ずオーナマシンに返送することで，最終的にはオーナマシンにおいて削除される必要がある．こうすることで，オーナマシンは厳密にブロックの位置情報とマスタートークンを保持するマシンの情報を管理しておくことが可能である．ブロックのアクセス要求があった場合の現在のブロックの位置を捜し出すオーバヘッドなどが低減される．

この場合，オーナマシンに様々なアクセスが集中することが問題点として考えられるが，現在のSLDの実装ではオーナマシンはブロック番号によってインタリーブされており，アクセスが各マシンに均等に配分されるようになっているので，ほとんど問題は起こらない．今後のSLDの拡張においても，アクセスの分散のために，オーナマシンを多数のマシンに分散することが重要であろう．

このマスタートークン方式では，マスターブロックがローカルなマシン上では厳密に全マシンのキャッシュ内で唯一のコピーブロックであるかどうかは判別できない．したがって，実際には無駄なファイルブロックの転送も行われる．しかし，次節で提案する優先度付きLRUアルゴリズムを置換アルゴリズムとして用い，ノンマスターブロックが積極的に置換され削除されることで，この無駄なファイルブロックの転送はほとんど発生せず問題にならない．

4.2 優先度付きLRUアルゴリズムによるブロック置換方式

SLDのキャッシュ管理方式において，あるブロックがリードアクセスされた場合に，オーナマシンがそのブロックのコピーを自キャッシュ内に保持しておき，他のマシンによるリードアクセスに備えている．この場合にマスタートークンはリードアクセス要求を出したマシンに渡されるので，オーナマシンのキャッシュ上にはノンマスターブロックが保持される．また，ファイルブロックが複数のマシンによってリードアクセスされる場合には，最初のマシンのみがマスターブロックを持ち，他のマシンはノンマスターブロックを保持する．

協調型キャッシュ全体の効率を考えた場合には，ノンマスターブロックがキャッシュ内を占有し，キャッシュ内のマスターブロックの置換を促進してしまうことは，処理能力の低下の原因となる．しかし，LRUアルゴリズムでは，通常はすべてのファイルブロックが置換対象として平等であることを前提にしている．そのために，上記の問題が発生する．つまり，協調型キャッシュ全体において貴重なマスターブロックと，協調型キャッシュ全体においてあまり貴重でないノンマスターブロックとを，平等に前回のアクセスからの時間間隔だけで置換対象としての評価を行うことに問題

がある．そこで，本論文では以下の優先度付きLRUアルゴリズムを提案している．

優先度付きLRUアルゴリズムでは，マスターブロックとノンマスターブロックを区別し，2つのLRUキューを用いてこれらを管理する．各ブロックに対して前回のアクセス時間を記録しておき，その前回のアクセス時間の順序でLRUキューにつながれる．置換対象のブロックを決定する場合には，2つのキューの先頭（最もアクセス時間が古い）のファイルブロックをとりだし，前回のアクセスからの時間間隔をある一定の重みを付けて比較し，重み付きの時間間隔の長いものを置換対象とする．マスターブロックにつける重みと比較して，ノンマスターブロックにつける重みを大きく設定することによって，マスターブロックは優先的にキャッシュ内に残され，ノンマスターブロックはより早く置換対象となる．つまりこの重みが各ブロックの種類に対する優先度となる．もちろん，協調型キャッシュでは，ノンマスターブロックは他のマシン上に少なくとも1つはファイルブロックのコピー（マスターブロック）があるので，このマシン上のブロックを即座に削除しても，次のアクセス時には他のマシンからファイルブロックを転送することで，そのファイルブロックへのアクセスのオーバヘッドは，再びオーナマシン上のディスクから読み込むよりも非常に小さい．

マスターブロックが置換対象となったときには，他のマシンのキャッシュへと転送するか，システム全体から見てそのファイルブロックが今後キャッシングしておく価値が低いと判断して削除するかを決定する必要がある．そのため，システム全体としてグローバルに，この優先度付きLRUアルゴリズムを近似する必要がある．つまり，あるマシン上でマスターブロックが置換対象となった時点で，以下のアルゴリズムを適用して，そのマスターブロックがシステム全体から見てキャッシングする価値が低いと判断された場合には削除される．

マシン i の2つのLRUキューの先頭のブロックのアクセス時間を，マスターブロックとノンマスターブロックそれぞれ T_i^m と T_i^n とする．現在の時刻を T としたときに，あるマシン a の置換対象のマスターブロックは次の条件式

$$T-T_a^m >= \max_{x\neq a}(\max(T-T_x^m, W(T-T_x^n))) \quad (1)$$

が成り立つ場合に削除される．ここで，W はノンマスターブロックの前回のアクセスからの時間間隔に対する重みであり，以降この W を（ノンマスターブロックの）優先度と呼ぶ．

この条件式(1)が成り立たないときには，次式

$$T-T_a^m < \max(T-T_y^m, W(T-T_y^n)) \quad (2)$$

が成り立つ少なくとも1台以上のマシン y が存在する．つまりマシン y には，マシン a 上の置換対象であるマスターブロックよりも，前回のアクセスからの時間間隔が長いマスターブロックか，もしくは重み付けした前回のアクセスからの時間間隔が長いノンマスターブロックが存在する．これらのブロックは，マシン a で置換対象であるマスターブロックよりも，協調型キャッシングのファイルシステム全体に対する価値が低いものであると判断されるので，マシン a からマスターブロックをマシン y に転送して，これらのブロックを削除するべきである．さらに，複数のマシンに対して上の条件式(2)が成り立つ場合には，それらのマシンで最大の $\max(T-T_i^m, W(T-T_i^n))$ の値を持つマシンに転送する．

4.3 実　　装

4.3.1 タイムスタンプ

上記の優先度付きLRUアルゴリズムを用いた協調型キャッシュを実装するために，各ファイルバッファのアクセス時のタイムスタンプを記録する必要がある．さらに，他のマシンでこのタイムスタンプを利用することから，各マシン上で時刻は同期をとっておくことが必要である．ただし，通常の使用において，各キャッシュブロックが置換されるまでの時間が数秒もしくは1秒以下であることは考えられないので，時刻同期の精度は秒単位で十分である．現在広く用いられているNTP（Network Time Protocol）などによる時刻同期で十分な精度が得られる．また，マスターブロックが他のマシンに転送されるときには，タイムスタンプ情報も同時に転送し，転送されたマシン上のLRUキューに挿入する際にも，そのタイムスタンプを利用してLRUキュー内のブロックの順序を保持する．

4.3.2 情報交換方式

分散してグローバルな優先度付きLRUアルゴリズムを実装するために，各マシンはLRUキューの先頭のブロックのタイムスタンプ（以降，LRUタイムと呼ぶ）の情報を他のマシンに転送する．すべてのメッセージには，この2つのLRUタイムを転送するための領域が設けられており，メッセージを転送する相手のマシンに自マシンの情報を転送する．また，ファイルブロックのアクセス要求をほとんど出さないアクティブでないマシンが存在する場合には，他のマシンとメッセージ交換が行われないため，そのマシンが保持する他のマシンに関する情報と，他のマシンが保持

するそのマシンに関する情報が古くなり，不正確なものとなる．そのために，各マシンで他のマシンに転送したLRUタイムを保持しておき，その保持したLRUタイムと実際のLRUタイムとのずれがある一定の閾値以上になった場合には，LRUタイム情報の交換のためのメッセージ転送を行う．この方法では，つねにメッセージにLRUタイム情報のための数バイトの情報が付加されるが，現在の広帯域ネットワークではほとんど影響がないと思われる．

4.3.3 グローバルな優先度付きLRUアルゴリズムの実装

SLDではマスタートークンはオーナマシンで管理される．オーナマシン以外のマシン上でマスターブロックが置換されるときには，必ずオーナマシンに転送される．そのために上記のグローバルな優先度付きLRUアルゴリズムを，各マシンの管理するブロックを置換するときだけに利用するように実装した．この方式は，オーナマシン以外のマシン上でマスターブロックが置換されたときに，オーナマシンが非常に負荷が高い場合にも，マスターブロックをオーナマシンに転送してしまう．そのため，マスターブロックを負荷の低いマシンに直接転送する場合に比べて，処理能力は低下する．しかし，マスタートークンやキャッシュコピーの位置情報などをオーナマシンが厳密に管理しやすく，実装が単純になるので，SLDではこの方式がとられている．

4.3.4 置換アルゴリズムの起動と停止

置換アルゴリズムの起動と停止には，通常の仮想記憶方式の置換アルゴリズムの起動方式として用いられる空きキャッシュブロック数に関する2つの閾値を利用する方式を採用している．まず起動時には，各マシン上で設定ファイルに従って固定数のキャッシュブロックが確保される．アプリケーションプログラムのアクセス要求に従い，このキャッシュブロックを利用して，ファイルブロックをキャッシュしてゆく．そして，空きキャッシュブロック数がある閾値（low free threshold）を下回ったときに，置換アルゴリズムを実行する置換スレッドが起動される．置換スレッドは，優先度付きLRUアルゴリズムに従って2つのLRUキュー上のファイルブロックを削除や転送することで空きキャッシュブロックを増やしていく．そして，空きキャッシュブロック数がもう1つの閾値（high free threshold）を上回った場合に置換スレッドは停止する．

4.3.5 SLDの内部構成

図3に示すように，SLDは大きくディレクトリ管理部とバッファ管理部，さらに今回拡張した情報管理部

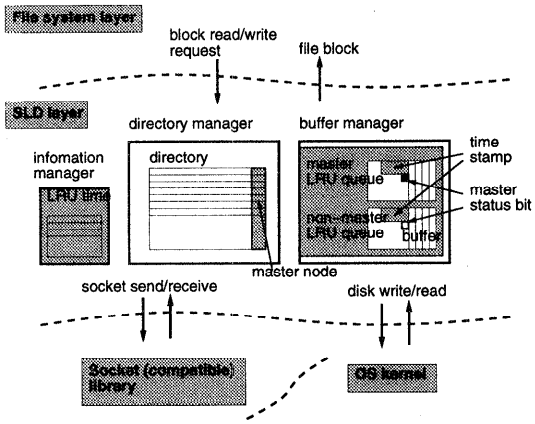


図3 協調型キャッシュアルゴリズムを実装したSLDの内部構成
Fig. 3 SLD structure with the cooperative caching mechanism.

に分離されている．ディレクトリ管理部で管理されるディレクトリには，そのマシンが所有するブロックのうち現在システム内のあるマシン上で利用されているブロックの情報が格納されている．今回は，このディレクトリを拡張してマストートークンを現在所有しているマシンの情報（マシンID）も格納する．

またバッファ管理部においては，そのマシン上のファイルシステム層が利用しているファイルバッファが管理されている．今回は，ファイルバッファのステータスとして，マストートークンの保有状況を示すステータスビットや，上記のタイムスタンプの情報を格納している．また，このバッファ管理部では各バッファがLRUキューにつながれているが，今回はここに優先度付きLRUアルゴリズムを実装するため，マスターブロック用とノンマスターブロック用の2つのLRUキューを用意している．

また，情報管理部では各マシンからメッセージとともに送られてくるLRUタイムを記録し，協調キャッシュとして利用するマシンを決定する際に利用する．

5. 性能評価

前章で提案した協調型キャッシュアルゴリズムの評価を行った．現在SLDはUnixやWindowsNT上で実装されており，Myrinetをネットワークの物理層として用いるソケットインタフェースの上に構築されている．今回の評価はOSとしてFreeBSD 2.2.6 RELEASEの動作した1台のPC/AT互換機☆上に，8個のSLDのサーバプログラムを起動することで，クラ

☆ プロセッサ：Intel Pentium 120 MHz，メモリ：Fast Page DRAM 64 MB，チップセット：Intel 430FX PCIset.

スタ型ファイルサーバのシミュレートを行った．

今回の評価では，実際のネットワークを利用せず同一のマシン上にあるプロセス間で通信を行っている．したがって，実際のファイルブロックのアクセス遅延の評価は不可能である．しかし，ネットワーク転送より非常に低速なディスクアクセス回数を抑えることでファイルブロックのアクセス遅延を低減できるという協調キャッシュの前提がある．今回は，この前提に従って，キャッシュヒット率を向上させディスクアクセス回数の低減することを目標として，協調型キャッシュアルゴリズムの評価を行った．

各サーバプログラムは自分の管理するファイルブロックを64K個持ち，システム全体の総利用ファイルブロック数は512K個である．また，各サーバプログラムは，4K個のキャッシュブロックを持ち，システム全体の総キャッシュブロック数は32K個である．また以下の評価動作を行うプログラムを記述し，ライブラリとして実現されている本協調型キャッシュアルゴリズムを実装したSLDをリンクして，テストプログラムを作成した．

評価テストの動作は以下のとおりである．まず評価テストに先立ち，全テストプログラムは利用可能ファイルブロックをランダムに10000回分の読み込みアクセスをサーバプログラムに要求することで，サーバプログラムのキャッシュブロックを埋める．その後，実際のテストフェーズに入る．まず，全サーバプログラムをアクティブなものとアクティブでないものに二分し，アクティブなサーバプログラムに対してだけ，ある大きさのワーキングセットに対してランダムに100000回のアクセスを行う☆．そのテストフェーズのキャッシュヒット率を計測する．テスト期間中は，アクティブでないサーバプログラムに対してはアクセス要求を出さないので，これらのサーバプログラムが協調型キャッシュの転送先になる．

5.1 従来のアルゴリズムとの比較

従来の研究にて紹介したアルゴリズムは，クライアント/サーバ型の分散ファイルシステムにおけるクライアントマシン間での協調型キャッシュアルゴリズムである．したがって，主にマスタートークンの扱いなどがSLDと異なる．そのため，SLDと対等に比較できるものではない．しかし，マスターブロックを他のマシンに転送するときの方針に関しては比較を行うことが可能である．今回は，マスターブロックが置換対象となったときにどのマシンに転送するかを決定するアルゴリズムに関して以下の4方式のアルゴリズムをSLDに実装し，本論文で述べた協調キャッシュアルゴリズムとの比較評価を行った．

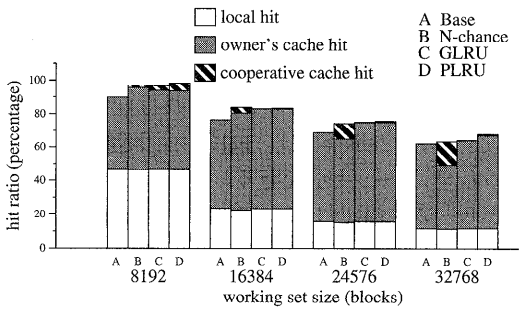


図4 各協調型キャッシュアルゴリズムでのヒット率の相違（アクティブプログラムが1つ存在する場合）

Fig. 4 Hit ratio for each cooperative caching algorithm with one active process.

Base 協調型キャッシュを用いない方式．

N-chance ランダムにマシンを選択し，N回転送☆☆した後は削除する方式．

GLRU グローバルなLRUアルゴリズムを利用する方式．

PLRU 優先度付きLRUアルゴリズムを利用する方式．優先度としては，式(1)，(2)の$W = 20$としている．

図4は，8個のサーバプログラムのうち，1つだけアクティブなサーバプログラムを動作させた場合のヒット率のグラフである．この場合には，アクティブでない残り7個のサーバプログラムが協調キャッシュとして利用されることになり，協調型キャッシュアルゴリズムによる協調キャッシュマシンの選択の効率度はあまり現れてこない．

全体のヒット率は，4方式のあまり違いは見られず，わずかではあるが優先度付きLRUアルゴリズムが最もヒット率が高い．ただし，N-chanceアルゴリズムだけは，協調キャッシュのヒット率がある程度大きい．N-chanceアルゴリズム以外のアルゴリズムの場合，協調キャッシュに転送されたマスターブロックのタイムスタンプを調べて，LRUキュー内のブロックのタイムスタンプの順序を保持するように挿入される．それに対して，N-chanceアルゴリズムでは，協調キャッシュに転送されたマスターブロックが，必ずLRUキューの先頭に挿入される．このために，N-chanceアルゴリズムでは，ブロックが置換されずに協調キャッシュ

☆ ここでは各プログラムがランダムに参照する個々の領域を，そのプログラムのワーキングセットと定義した．

☆☆ ここでは$N = 1$とした．

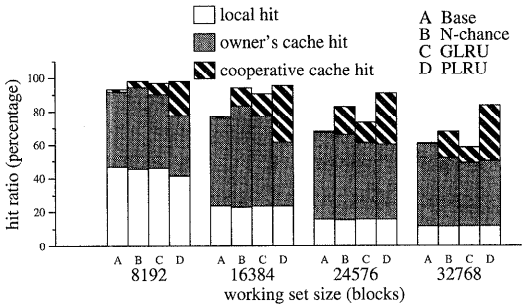


図5 各協調型キャッシュアルゴリズムでのヒット率の相違（アクティブプログラムが4個存在する場合，ワーキングセットは各プログラムで同一）

Fig. 5 Hit ratio for each cooperative caching algorithm with four active processes which have the overlapped working set.

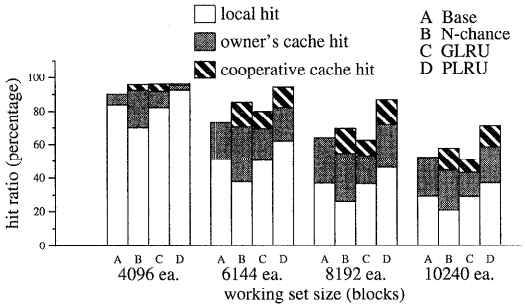


図6 各協調型キャッシュアルゴリズムでのヒット率の相違（アクティブプログラムが4個存在する場合，ワーキングセットは各プログラムで独立）

Fig. 6 Hit ratio for each cooperative caching algorithm with four active processes which have the distinct (non-overlapped) working set.

表1 各協調型キャッシュアルゴリズムでのディスクアクセス回数（アクティブプログラムが4個存在する場合，ワーキングセットは各プログラムで同一）

Table 1 Disk access requests for each cooperative caching algorithm with four active processes which have overlapped working set.

ワーキングセットサイズ	Base	N-chance	GLRU	PLRU
8192	27017	7967	12332	7804
16384	91646	23663	37753	18568
24576	128140	67844	106417	36488
32768	155787	128297	165508	65109

表2 各協調型キャッシュアルゴリズムでのディスクアクセス回数（アクティブプログラムが4個存在する場合，ワーキングセットは各プログラムで独立）

Table 2 Disk access requests for each cooperative caching algorithm with four active processes which have non-overlapped working set.

ワーキングセットサイズ	Base	N-chance	GLRU	PLRU
4096	39528	17321	16099	15737
6144	108060	58811	80606	23763
8192	144209	122526	150812	53067
10240	191837	169861	196949	116410

に存在する時間が長く，協調キャッシュ上でのヒット率は大きくなる．

図5 に，8個のサーバプログラムのうち，同じワーキングセットをアクセスする4個のアクティブなサーバプログラムを動作させた場合のヒット率を示す．この場合，アクティブな4個のサーバプログラムから，アクティブでない4個のサーバプログラムが協調キャッシュとして利用されることになり，各アルゴリズムの協調キャッシュとして利用するマシンの選択方式の相違が大きくヒット率に影響する．また，**表1** にディスクアクセス回数を示す．

全体のヒット率に関しては，優先度付きLRUアルゴリズムが最もヒット率が高い．特にワーキングセットが大きく，協調型キャッシュを頻繁に利用する状況下で，優先度付きLRUアルゴリズムは他のアルゴリズムに比較して最大で23%ヒット率が改善され，またディスクアクセス回数は半減している．

しかし，協調キャッシュのヒット率が，他のアルゴリズムと比較して大きく，ファイルブロックの平均アクセス遅延は全体のヒット率の違いほど大きくは改善されないという可能性もある．これに関しては，実機上での平均アクセス遅延の計測を行い評価してみる必要があるが，Myrinetのような広帯域・低遅延のネットワークデバイスを用いた場合には，ディスクアクセス回数の低減の効果は大きく，平均アクセス遅延も本図の全体のヒット率と同様の傾向になると予想される．

図6 に，8個のサーバプログラムのうち，各々独立したワーキングセットをアクセスする4個のアクティブなサーバプログラムを動作させた場合のヒット率を示す．この場合も上の場合と同様に，アクティブな4個のサーバプログラムから，アクティブでない4個のサーバプログラムが協調キャッシュとして利用されることになり，各アルゴリズムの協調キャッシュとして利用するマシンの選択方式が大きくヒット率に影響する．また，**表2** にディスクアクセス回数を示す．

この場合も，全体のヒット率に関しては優先度付きLRUアルゴリズムが最もヒット率が高く，特にワーキングセットが大きく協調型キャッシュを頻繁に利用する状況下で，優先度付きLRUアルゴリズムは他のアルゴリズムに比較して最大で25%ヒット率が高くなる．またディスクアクセス回数は最大で他のアルゴリズムの60%の低減が可能である．さらに，この場合に

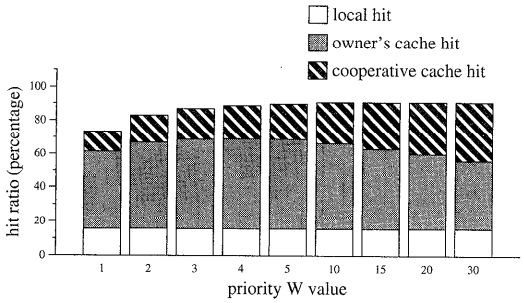


図7 優先度を変化させた場合のヒット率（ワーキングセットは各プログラムで同一）

Fig. 7 Hit ratio vs. various priority parameter W with four active processes which have overlapped working set.

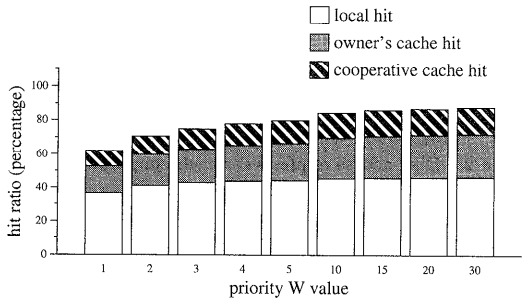


図8 優先度を変化させた場合のヒット率（ワーキングセットは各プログラムで独立）

Fig. 8 Hit ratio vs. various priority parameter W with four active processes which have non-overlapped working set.

は上記の同じワーキングセットを用いていた場合と異なり，優先度付きLRUアルゴリズムのローカルキャッシュのヒット率も，他のアルゴリズムと比較して最も高いので，ファイルブロックの平均アクセス遅延はより大きく低減される．

5.2 優先度に関する評価

優先度付きLRUアルゴリズムに関して，優先度を変化させた場合の評価も行った．**図7**は，8個のサーバプログラムのうち，同一のワーキングセットを持つ4個のアクティブなプログラムを動作させた場合の，式(1)，(2)の優先度 W を変化させたときのヒット率の変化を示したものである．

この場合，全体のヒット率は優先度を5以上にした場合にほぼ一定になり，逆にローカルキャッシュのヒット率は優先度を5以上にした場合には徐々に低下する．全体のヒット率が変化せずに，ローカルキャッシュのヒット率が下がれば，ファイルブロックの平均アクセス遅延が大きくなるので，この場合の優先度は5が最適であるといえる．優先度をあげることは，マスターコピー以外のファイルブロックを早期に置換してしまうことにつながり，そのためローカルキャッシュのヒット率が低下してしまう結果となり，平均アクセス遅延を低下させる原因になる．

さらに，8個のサーバプログラムのうち，独立したワーキングセットを持つ4個のアクティブなプログラムを動作させた場合の，式(1)，(2)の優先度 W を変化させたときのヒット率の変化を**図8**に示す．

この場合には，かなり大きな優先度になっても全体のヒット率は向上し続け，さらにローカルキャッシュへのヒット率も同様に向上している．つまり，ワーキングセットが独立しており，マスターコピーが多くを占める場合には優先度をかなり大きくとることによって性能が向上することが分かる．

この優先度に関する2つの評価では，すべてのワーキングセットが独立である場合と，すべてのワーキングセットが同一である場合という両極端についての評価を行った．実際のアプリケーションでは，ワーキングセットがまったく同一であったり，まったく独立であることはなく，その中間的なものになる．つまり，今回の実装やサーバプログラムの負荷に関して，優先度の最適値は5より大きく10から20といったあたりに存在すると考えられる．

6. おわりに

本論文では，クラスタ型分散ファイルシステムにおける協調型キャッシュアルゴリズムに関して論じ，クラスタ型の分散ファイルシステムにおけるキャッシュの一貫性管理機構に組み込まれた協調型キャッシュアルゴリズムを提案した．本協調型キャッシュアルゴリズムでは，マスタートークンを用いてブロックを管理し，ノンマスターブロックを優先的に置換する方式である優先度付きLRUアルゴリズムを利用している．また，本アルゴリズムの実装方式，および評価結果を述べた．評価結果として，優先度付きLRUアルゴリズムは，比較に用いたN-chanceアルゴリズムや，グローバルなLRUアルゴリズムよりも，ディスクアクセス回数を最大で60%低減可能であり，最大でヒット率を25%近く改善できるという結果を得た．これによりファイルの実際の平均アクセス遅延も十分低減可能であると考えられ，処理能力の高いクラスタ型分散ファイルシステムが構築可能となる．

今後の課題として，ファイルブロックの平均アクセス遅延の計測による評価を行うこと，実際のファイルシステムを動作させ，NFSサーバなどのアプリケーションによる評価を行う必要がある．

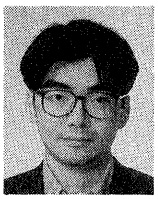
謝辞 本研究を進めるにあたりご指導いただいたプ

リンストン大学のKai Li教授，Edward Felten助教授，Rober Shillner氏に感謝する．また，有益な助言をいただいた情報メディア研究所柴山茂樹所長，分散メディア第一研究室吉本雅彦室長に感謝の意を表する．

参考文献

1) Anderson, T.E., Dahlin, M.D., Neefe, J.M., Patterson, D.A., Roselli, D.S. and Wang, R.Y.: Serverless Network File Systems, *Proc. 15th ACM Symposium on Operating Systems Principles*, pp.109–126 (1995).
2) Dahlin, M.D., Wang, R.Y., Anderson, T.E. and Patterson, D.A.: Cooperative caching: Using remote client memory to improve file system performance, *Proc. 1st USENIX Symposium on Operating Systems Design and Implementation*, pp.267–280 (1994).
3) Feeley, M.J., Morgan, W.E., Pighin, F.H., Karlin, A.R. and Levy, H.M.: Implementing Global Memory Management in a Workstation Cluster, *Proc. 15th ACM Symposium on Operating Systems Principles*, pp.201–212 (1995).
4) Hartman, J. H. and Ousterhout, J. K.: The Zebra Striped Network File System, *Proc. 14th ACM Symposium on Operating Systems Principles*, pp.29–43 (1993).
5) Lee, E.K. and Thekkath, C.A.: Petal: Distibuted Virtual Disks, *Proc. 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.84–92 (1996).
6) Rosenblum, M.and Ousterhout, J.K.: The Design and Implementation of a Log-Structured File System, *Proc. 13th ACM Symposium on Operating Systems Principles*, pp.1–15 (1991).
7) Sarkar, P. and Hartman, J.: Efficient Cooperative Caching using Hints, *Proc. 2nd USENIX Symposium on Operating Systems Design and Implementation*, pp.35–46 (1996).
8) Shillner, R.A. and Felten, E.W.: Simplifying Distibuted File Systems Using a Shared Logical Disk, Technical Report, TR-524-96, Department of Computer Science, Princeton University (1996).
9) Thekkath, C.A., Mann, T. and Lee, E.K.: Frangipani: A Scalable Distibuted File System, *Proc. 16th ACM Symposium on Operating Systems Principles*, pp.224–237 (1997).
10) Wand, R.Y. and Anderson, T.E.: xFS: A WIde Area Mass Storegae File System, *4th Workshop on Workstation Operating Systems*, pp.71–78 (1993).

(平成10年12月4日受付)
(平成11年 4 月1日採録)



数藤 義明（正会員）

1989年東京大学工学部計数工学科卒業．1991年同大学大学院工学系研究科情報工学専攻修士課程修了．同年キヤノン（株）入社．1996年より2年間，プリンストン大学コンピュータサイエンス学科客員研究員．並列・分散オペレーティングシステムの研究開発に従事．並列・分散オペレーティングシステム，計算機アーキテクチャ，並列・分散処理に興味を持つ．ACM，IEEE CS各会員．