社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－OS－100 (4)
2005／8／3

キャッシュサーバを用いた大規模分散ファイルシステムとその応用

小柳順裕[†] 田胡和哉[†] 山下直人[†]

兵頭和樹[†] 松下 温[†]

大規模なネットワーク環境における情報共有インフラの必要性が高まっている。その試みの一つとして、NFSファイルシステムにキャッシュ機能を提供するノードを追加し、多階層構造とすることによってスケーラビリティを改善する試みを紹介する。このシステムを、Community Storage と名づけた。キャッシュノードでは、キャッシュ記憶媒体とネットワークを効率的に利用するためのスケジューリングを行い、自動的な性能向上を図る。現在（2005年7月）、サブセット機能版の Ver.0 のシステムテストを行っており、Diskless クライアントの立ち上げ、および、eclipse が動作することを確認した。またこれと平行して Ver.1 の開発と、実証実験の準備をすすめている。

Design and Implementation of a Large-scale Distributed File System Using Cache Servers

MASAHIRO KOYANAGI,[†] KAZUYA TAGO,[†] NAOTO YAMASHITA[†]

KAZUKI HYODO[†] and YUTAKA MATSUSHITA[†]

The necessity of the infrastructure for information sharing in a large-scale network environment is rising. As one of the attempts, this paper introduces an attempt to improve the scalability by composing a multilayer structure adding the nodes that provide the cash function to the NFS. This system was named Community Storage. The cash node aims an automatic performance improvement of the efficient use of the cash storage media and the network by scheduling. The system test of Ver.0 that is the subset function version is running, and it was confirmed that the Diskless Client boot upped, and the Eclipse operated now (July, 2005). Also, we are developing Ver.1 and are preparing the demonstration experiment in parallel.

研究は、文部科学省 私学高度化助成 オープンリサーチセンタ「Linux オープンソースソフトウェアセンタ」によって実施されている。

東京工科大学
Tokyo University of Technology

1. はじめに

ネットワーク技術の発展が著しい。たとえば、従来からのLANにおいても、1万台をこえる計算機を接続した構内ネットワークを構成することが比較的容易に行えるようになっている。

これに対応して、大規模なネットワーク環境における情報共有インフラの必要性が高まっている。そのような試みの一つとして、単純なクライアントサーバ構成を持つNFSファイルシステム[1]にキャッシュ機能を提供するノードを追加し、多階層構造とすることによってスケーラビリティを改善する試みを紹介する。このような、NFSサーバとキャッシュノードからなるシステムを、Community Storageと名づけた。

Community Storageは、企業、公共組織、学校等、大規模組織において、構成員全員が同一の条件で利用できる情報共有機構を提供するばかりでなく、ネットワークブートによるDisklessクライアントの実現、マルチメディアコンテンツの共有にも利用できる。

Community Storageの実現上の大きな目標は、キャッシュの記憶媒体やネットワークに関して、自動的な性能チューニング機能を実現することにある。ここでは、その設計方針、および、実現の現状について述べる。

2. システムの概要

図1に、ここで実現の対象としているCommunity Storageシステムの全体構造を示す。ファイルサーバノードと、キャッシュノードが分散ファイルシステムとしてのサービスを提供し、これにクライアントノードが接続されている。クライアントノードからは、1台のファイルサーバと多数のキャッシュノードからなるシステム全体が1台のNFSファイルサーバであるかのように見える。

ここでは、最大1000台程度のキャッシュノードを1台のファイルサーバに接続することを想定している。これによって、1万台を超えるクライアントノードを接続することができる。キャッシュノードは、メモリのみならず、ディスク装置や他の記憶装置を記憶媒体としてファイルキャッシュ機能を実現する。キャッシュノードは、ファイルサーバノードに対してはNFSクライアントとして動作し、クライアントノードに対してはNFSサーバとして動作する。ファイルサーバからインポートしたファイルを、キャッシュ記憶媒体に一時保管し、さらにクライアントノードに対してエクスポートする。ファイルシステムを利用したWebプロキシや、Sambaシステムをキャッシュノード上で動作させることにより、Windows PC間でのデータ共有や、Webコンテンツの共有にも拡張できる。

ファイルサーバノードとキャッシュノードの間はNFS V4プロトコルによって接続し、Delegation/Recall機構によってキャッシュコヒーレンシを実現する。さらに、大容量ファイルの共有効率を改善するために、Delegation/Recall機構を拡張する。

サーバノードは、Delegation/Recallの制御を行うとともに、バックアップサーバとして動作する。キャッシュノードに書き込まれたファイルは、比較的長期

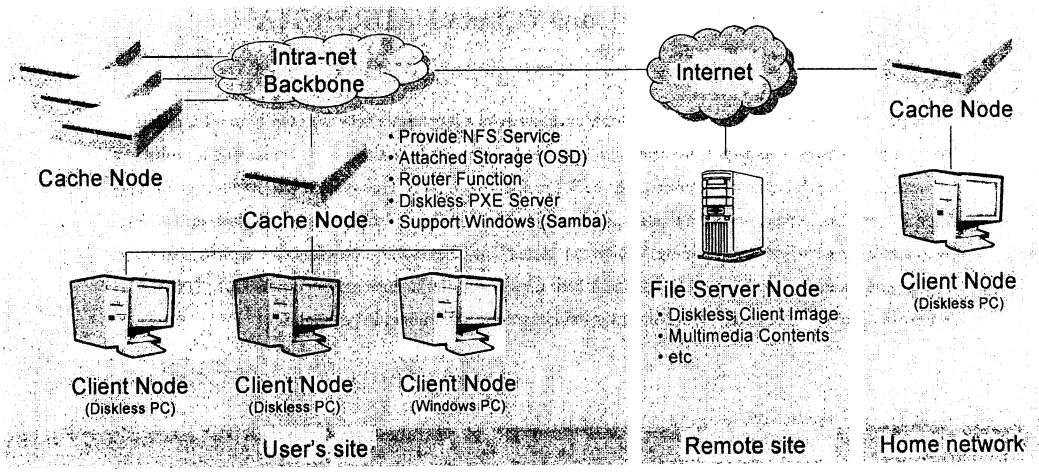

図1 システムの全体構造

間キャッシュノード中に滞在し、ネットワークに過度の負荷をかけないことを確認しながらサーバノードにライトバックされる。サーバノードには最終的にはすべてのファイルが転送され、バックアップされることを想定する。

キャッシュノードは、ルータ機能を兼ね備え、無線 LAN のアクセスポイントとしても動作する。現状では、PC を用いて運用しているが、最終的にはネットワークプロセッサを用いることを検討している。キャッシュノードは、家庭に 1 台設置したり、オフィスの各部屋に 1 台ずつ設置したりする。

キャッシュノードには、ストレージが付加される。ストレージアクセスのためのプロトコルとして Object-Based Storage Device (OSD) [2] を用いる。OSD を用いることによって、少ないオーバヘッドでスケーラブルな性能を持つキャッシュノード向きストレージデバイスを実現することができる。現在、キャッシュノード機能のネットワークプロセッサへの移行、および、USB によって接続できる小型の OSD 装置の開発を検討している。

クライアントノードに対しては、通常のファイル共有サービスを提供する。なかでも特に、

1) ネットワークブートする Diskless クライアントのファイルサーバ
2) マルチメディアコンテンツの共有

に利用することを想定している。

ネットワークブートする Diskless クライアントは、ハードウェア上は通常の PC であるが、ハードディスクを持たない。これによって、

1) 管理の容易化
2) セキュリティの向上

を図る。システム全体で、一つのクライアントシステムイメージを共有することによって、ソフトウェアの管理が著しく容易化され、ひいては運用コストの軽減を図ることができる。また、クライアントハードウェア上にパーシステントなデータを持たないことによって、データが不正に持ち出される機会を減らすことができる。

Java を用いた分散データベース管理システムを利用してメタデータの分散共有機構を実現し、これと Community Storage を組み合わせて利用することにより、マルチメディアコンテンツの共有機構を構築する。Java によるメタデータ管理機構と組み合わせることにより、Java の API から、ネットワーク上のすべてのメタデータ付きのコンテンツを、あたかもローカルディスクにあるようにアクセスできるようになる。これによって、たとえば、マルチメディアを自由に利用できるグループウェアが実現できる。

3. アーキテクチャ

ファイルサーバノードとクライアントノードの構造は、通常のシステムと変わらない。ここでは、2. で述べた機能を実現するキャッシュノードの内部構造について述べる。

図 2 に、キャッシュノードの内部構造を示す。キャッシュノードは、Linux で実現されており、これに、キャッシュノード用のカーネルモジュールと、ユーザモードで動作するデーモン (Scheduler) を追加することによって構成される。両者は、仮想デバイスを利用したメッセージング機能を利用して連携して動作する。

キャッシュノード用のカーネルモジュールは、Data Path Switch (DPS) とよばれる。これは、

1) 個々のファイルごとに、キャッシュ用のファイルを割り付け、これを利用してキャッシュ動作を行う
2) サーバノードのファイルを NFS プロトコルによってインポートした後に、再度 NFS プロトコルによってエクスポートする

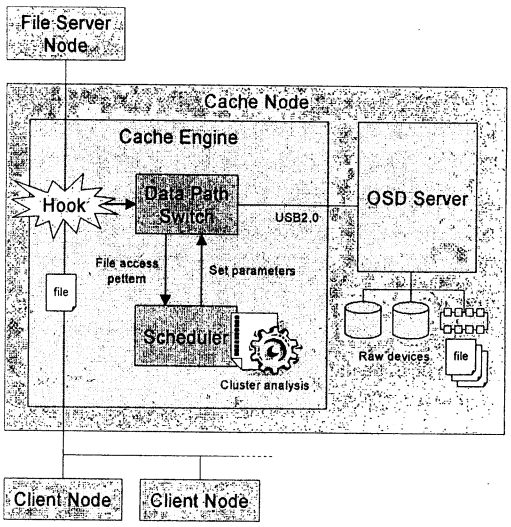

図 2 キャッシュノードの内部構造

3) クライアントノードからのファイルアクセスパターンのログを取得する

機能を持つ。ファイルごとのキャッシュ媒体も、それぞれ別個のファイルを用いるので、キャッシュ用の記憶媒体として、主記憶上のファイルシステム（RAM ファイルシステム）、通常のファイルシステム、OSD 等を自由に選択することができる。DPS は、キャッシュ動作中にキャッシュ記憶媒体を変更する機能、サーバノードへのライトバックの速度を調節する機能、サーバノードからのプリフェッチの速度を調節する機能、ライトバックの際の、キャッシュイメージのバージョンを管理する等を持つ。また、NFS V4 プロトコルで規定されている Delegation/Recall 機能に対応したキャッシュのライトバック、無効化機能を持つ。

キャッシュノード用のデーモンは、Scheduler とよばれる。これは、

1) DPS が提供するアクセスパターンのログの分析を行い、ファイルごとのキャッシング戦略を決定する。
2) 決定されたキャッシュ戦略にもとづいて、DPS に対して、ファイルごとに、キャッシュ媒体の選択やプリフェッチ、ライトバックの速度の指定を行う。

Scheduler は、C++言語によるオブジェクトフレームワークの形式をとる。これによって、種々のキャッシング戦略を、必要に応じて容易に追加できる。

DPS は、1 秒に一度ずつ、キャッシュ対象となっているファイルのログ情報を Scheduler に報告する。Scheduler は、これにもとづいてキャッシュ戦略の策定を行い、決定事項を DPS に伝達する。このように、DPS と Scheduler の間の通信をタイマ駆動とすることにより、通信の機会を著しく削減することができ、オーバヘッドの軽減につながる。一方において、Scheduler をユーザプログラムとすることによって、開発の容易化を図ることができる。

カーネル内部のモジュールとデーモンを用いる構造は、たとえば、Coda ファイルシステム[3] のアーキテクチャと類似しているが、使用目的は以下の点において大きく異なる。

1) デーモンはファイルデータのやりとりに直接参加しない。Coda では、ファイルレプリカがローカルに存在しない場合はデーモンがそのフェッチを行っているが、Community Storage ではフェッチのスケジューリングのみを行う。
2) デーモンは、非同期的に動作する。Coda では、データパスの構造から必然的に、同期的に動作する必要がある。すなわち、ファイルレプリカがローカルに存在しない場合には、デーモンに処理を依頼し、それが完了するまではカーネル部分はそのファイルに関する処理を再開できない。Community Storage では、ファイルアクセスに関する同期的な処理はすべてカーネルモジュールが実行し、Scheduler は性能改善等の目的で非同期的に介入するだけである。

以上より、キャッシュノードの動作シナリオは以下のようになる。Scheduler は、はじめてキャッシュされるファイルに関して、キャッシュ記憶媒体の選択、ライトバック速度等に関するパラメータを、あらかじめ DPS に伝達しておく。これを、デフォルトキャッシュ戦略とよぶことにする。DPS は、動作をはじめると、Scheduler の助けなしに、デフォルトキャッシュ戦略にもとづいてキャッシュ動作を開始する。

Scheduler は、定期的にファイルアクセス状況を観測し、繰り返しアクセスされるファイルに対しては、個々にキャッシュ戦略を変更する。DPS は、キャッシュ戦略の変更に対応して、個々のファイルごとに、プリフェッチ速度、ライトバック速度、キャッシュ記憶媒体を実行時に変更できる機能を持つ。

ファイルキャッシュに用いる記憶媒体として、ハードディスクによる通常のファイルシステムのみならず、OSD も用いることができるようにする。OSD は、記憶装置にファイルシステム機能の一部をオフロードし、セクタ単位ではなく、ファイル単位で記憶装置にアクセスするための外部記憶アクセス規約である。通常では、iSCSI[4] 等の、ストレージネットワークに用いることを想定しているが、キャッシュノードの実現にもよく適合する。たとえば、以下のような利用方法が考えられる。

キャッシュノードの実現方法のひとつとして、ネットワークプロセッサを用いたルータ装置を利用することがあげられる。最近の小型ルータ装置には、USB インタフェースを持つものが多く、これを利用して外部にディスク装置を付加することができる。しかしながら、このようなルータ装置には種類が少なく、性能上のスケーラビリティを得ることが難しい。そこで、高い性能が必要な場合には、外部にディスク装置のみならずメモリやプロセッサを付加して対応することを考える。このような外部記憶装置とルータ装置を接続

する方法として、ATA 等のディスクアクセスプロトコルをそのまま用いると、メタデータアクセスの際にもディスクアクセスを行う必要が生じ、トラフィックが増大する危険がある。この問題を避ける方法として、OSD プロトコルは有効である。OSD プロトコルによれば、ファイル単位でのアクセスが可能になるために、メタデータアクセスの頻度を削減することができる。さらに、キャッシュノードでは、アクセス権のチェックは別の機構を用いて行っているために、OSD の機能との整合性が高く、きわめて効率の高いファイルキャッシュのための外部記憶機構を実現することができる。

4. キャッシュスケジューリング方式

4.1 キャッシュスケジューリングの概要

キャッシュノードにおけるキャッシュ管理について、以下の２点の観点からのスケジューリングが必要になる。

1) キャッシュ記憶媒体の効率的利用
2) ネットワークの効率的利用

Community Storage では、キャッシュ記憶媒体として、メモリ、および、ディスク装置を主に利用することを想定している。したがって、

1) メモリとディスク装置の使い分けの決定
2) ディスク装置の効率的利用

に関してスケジューリングを行う必要が生ずる。これが、キャッシュ記憶媒体の効率的利用に相当する。

ネットワークの規模が大きくなると、特定のネットワークリンクに負荷が集中してネットワークが破綻する危険が増大する。したがって、システムがネットワーク全体の負荷状況を把握して、トラフィックを管理する機能が不可欠になる。Community Storage の狙いのひとつは、マルチメディアコンテンツの収集、配信系も兼ねることにある。したがって、通常ファイルに関する分散アクセスの管理と、大規模なマルチメディアファイルの分散アクセスの管理の双方が実現されている必要がある。

ファイルサーバノードは、ファイルのバックアップを集中して実行する機能を提供しており、ファイルサーバノードへのライトバックは不可欠である。その一方において、Community Storage では、ネットワーク中に多数のキャッシュノードを分散して配置する。このような系においては、書き込みのあったファイルを頻繁にファイルサーバノードにライトバックすることは効率的でない。したがって、

1) ファイルサーバノードへのライトバックのタイミング
2) キャッシュノード間の連携によるネットワークトラフィックの軽減

に関してスケジューリングを行う必要が生ずる。たとえば、ファイルサーバノードへのライトバックは、ネットワークトラフィックやファイルサーバノードの負荷状況を監視して、負荷の少ない夜間に実行することが考えられる。また、キャッシュ間連携を、キャッシュ記憶の実行容量の増大のためではなく、ネットワークトラフィックの最適化のために利用することができる。すなわち、キャッシュノードにある書き込みファイルイメージをファイルサーバノードにライトバックすることなく、他のキャッシュノードに移動させることを考える。これにより、マルチメディアの配信系における、複数キャッシュ間での連携動作と同様のモードを包含したキャッシュ運用が可能になる。これらのスケジューリングがネットワークの効率的利用のためのスケジューリングに相当する。

4.2 キャッシュ記憶媒体の効率的利用に関するスケジューリング

ディスク装置をキャッシュ記憶媒体として利用する際の性能上の考慮点は、通常のファイルシステムを構築する際の考慮点と類似するものも多いが、いくつかの重要な点において異なっている。ここでは、キャッシュノードのキャッシュ記憶媒体に用いるための専用のストレージシステムを構築することを前提として、その設計について述べる。以下では、このストレージシステムを、キャッシュ記憶システムとよぶことにする。キャッシュ記憶システムは、キャッシュノード内で実装することも可能であるが、現状では、OSD プロトコルを用いてキャッシュノードの外部に付加することを考えている。

キャッシュ記憶システムの動作の前提となる条件は、通常のファイルシステムのそれと以下の点において異なっている。

1) read キャッシュの場合は、キャッシュ記憶媒体に書き込む時点でファイルサイズがあらかじめ判明している。
2) 記憶容量の利用効率は高い必要がない。

3) read キャッシュは、場合によって放棄することができる。

このような点を考慮し、キャッシュ記憶媒体として利用するための専用のストレージシステムを構築することにする。たとえば、記憶媒体のフラグメンテーションの可能性が増大しても、物理入出力効率が高いことが望ましい。そこで、おもな設計方針として、

1) ディスク領域の割り当て単位を、従来のファイルシステムにおけるそれよりはるかに大きく、１０Ｍバイト程度とする
2) ファイルやファイル集合に対して連続領域を割り当てる
3) ファイルアクセスパターンを解析し、クラスタ分析によって相互に関連のあるファイルから構成されるファイル集合を決定する
4) メモリの総容量によってディスク入出力の単位サイズを変更する

等をとる。ここでの、ディスク領域の割り当て単位をクラスタとよぶことにする。キャッシュすべきファイルのサイズによって、単一のクラスタに複数のファイルが収容される場合と、複数のクラスタによって単一のファイルが構成される場合が生ずる。

クラスタは、Scheduler によるファイルアクセスに関する統計処理と深く結びついている。システムのブート時や、コンパイル処理を行う場合に典型的にみられるように、単一のユーザ処理を行う場合でも、複数のファイルが必要とされる場合が多い。このような、ファイル同士が連続してアクセスされる頻度に関する統計的な分析から、ファイル間の“距離”に関する定義が可能になる。これを、アクセス距離とよぶことにする。アクセス距離の小さいファイル同士は、短い時間内に同時にアクセスされる確率が高い。したがって、アクセス距離の小さいファイルを単一のクラスタ内に収容し、同時にメモリ上にフェッチすれば、物理入出力のコストを軽減することができる。このとき、クラスタをメモリ上にフェッチすることを、クラスタステージングとよぶことにする。

キャッシュ記憶システムには、大容量のメモリを搭載することを想定している。ファイルは、一度オープンされると全体が read/write される傾向が大きいことから、このメモリをページ単位で置き換えるページキャッシュシステムとして動作させるのではなく、クラスタステージングのための領域として利用する。

ディスク領域の割り当てスケジューリングは、クラスタに対するファイル割り当てと、ディスクに対するクラスタ割り当てに分割されることになる。たとえば、複数のクラスタを必要とする大きなファイルを収容する場合には、できるだけ連続したクラスタが割り当てられることが望ましい。

ファイルアクセス特性から考えて、クラスタに対するファイル割り当てが全体の性能を大きく左右することが予想される。長期間にわたるファイルアクセス特性の統計的解析と、バックグラウンドジョブによるファイルのクラスタへの再割り当て機構を実現することによって、自動的なシステム性能の改善機構を実現する。

4.3 ネットワークの効率的利用に関するスケジューリング

NFS の本来の設計では、ファイル書き込みによるダーティデータは、最終的には、ファイルサーバノードにライトバックされる。しかしながら、たとえば、マルチメディアコンテンツファイル等の大容量ファイルは、単純にファイルサーバノードにライトバックするのではなく、キャッシュノード間で相互に転送する方式の方がネットワークの利用効率が改善されることが予想される。したがって、大容量のファイルのライトバックでは、ネットワークの負荷状況を観測し、ファイルの移動先を動的に変更する方式をとる。ファイルサーバノードは、ネットワーク中のファイルの存在位置を把握しているので、機構上は、このような移動は比較的容易に実現できる。

ファイルの移動先を決定する方法について述べる。ファイルの移動先のスケジューリングは、ネットワークのトポロジ、バンド幅、負荷状況等に関する詳細な情報を必要とする。したがって、最初から最適に実施することは期待できない。スケジューリング過程自体に、試行と評価のプロセスを含んでいる必要がある。

分散環境におけるキャッシュの管理方式として、キャッシュ間連携があげられる。キャッシュノード間でなるべく異なるファイルをキャッシュすることによって、ネットワーク全体としてのキャッシュ記憶媒体の利用効率を高めようとする方式である。その一方において、キャッシュ間連携によってファイルアクセスのネットワーク局所性が低下する恐れがある。すなわち、遠方のキャッシュノードにまでキャッシュをとりに行かなければならない可能性が増大する。そこで、試行においてキャッシュ間連携のスコープを決めることが必要である。近隣のキャッシュノード間ではキャ

ッシュ間連携によって重複してファイルを持つのを避け、遠隔のキャッシュノード間ではキャッシュイメージの複製を持つようにする。

試行方式について述べる。複数のキャッシュノードからなるセルを定義し、同一のセル内ではファイルの移動を行わないようにする。セルの構成は動的に変更される。複数のセルを融合して一つの新たなセルを定義し、運用を行う。この結果、ネットワーク負荷の低下、および、クライアントへのサービスレベルの向上がみられなければセルを解消して、別のセルを構成する。このような、セルの再構築を通じて、ネットワーク全体での自動的な性能改善を図る。

5. 現状

5.1 実装

原稿執筆時（2005年7月）に、サブセット機能版のVer.0が完成し、システムテストを行っている。カーネルモジュール部分（DPS）はほぼ全機能が実装されている。また、Schedulerのオブジェクトフレームワークが実装されている。Ver.0のキャッシュノードは、Linuxのディストリビューションの一つである、FedoraCore2（以下FC2）を用いている。サーバノードとキャッシュノードの間は、NFS V3,V4双方を利用することができるが、Linux用NFSの実装自体にDelegation/Recallが含まれていないために、コマンドによってキャッシュノード間のキャッシュコヒーレンシを制御している。Ver.0のSchedulerは、単純なアルゴリズムで構成され、クラスタ分析機能は実装されていない。

現時点で、Ver.0を用いることにより、Disklessクライアントの立ち上げ、および、アプリケーションの実行ができるようになっている。また、eclipseがDisklessクライアント上で動作することを確認した。

Ver.0では、キャッシュノードの実現に加えて、OSDの実現を行っている。これまで、OSDは、iSCSI、もしくは、ファイバチャネルプロトコルを利用して実装されてきた。ここでは、ストレージネットワークを実現することは目的ではないので、OSDプロトコルを直接TCP/IP上で実装することによって、キャッシュノードとOSD記憶装置を1対1で結合する方法をとっている。この実装は、少ない変更で通信媒体としてUSBを利用する方式にも適用できるように配慮されている。

OSD記憶装置自体もLinuxで実現され、現状では、OSDプロトコルを解釈実行するエンジンはアプリケーションプログラムとして実装されている。キャッシュノードの記憶媒体としては通常のファイルシステムを使用しており、OSDを用いたキャッシュ記憶システムは現在実現作業中である。現状で、OSDプロトコルの実装をほぼ完了した。

Ver.0のテストと並行して、Ver.1の開発を行っている。Ver.1では、FedoraCora4以降のディストリビューションを用い、Delegation/Recall機構によるキャッシュコヒーレンシ機構を実現する予定である。これによって、実証実験が行えるようになることが期待される。

最近ネットワークプロセッサを用いたボードが市販されるようになっている。これは、Linuxで動作するので、Community Storageが比較的容易に移行できることが期待される。この環境での実装も検討している。これを、キャッシュルータと名づけることにする。

5.2 DPSの実装

Data Path Switch（DPS）の実装について述べる。DPSは、独立したカーネルモジュールとして実装され、約6000行のソースコードからなる。Linuxのカーネルや NFSの実装が並行して進められているので、これらの実装を変更せずにキャッシュノード動作が実現できるように配慮されている。

DPSは、起動すると、キャッシュ動作の対象となるファイルシステムのスーパブロック、および、すべてのinodeの操作関数表（super_operations, file_operations, inode_operations）を書き換えることによって、対象となるファイルシステムの動作を変更する。ここでは、ファイルサーバノードがエクスポートするファイルをキャッシュノードにインポートするためのNFSクライアントファイルシステムが書き換えの対象となる。これによって、ファイルサーバノードに対するファイルアクセス処理や、inodeの変更処理はすべてフックされ、DPSが動作に介入できるようになる。

DPSは、ファイルごとのキャッシュ動作を管理するための表を保持しており、これを用いてキャッシュ動作を実現する。キャッシュ記憶媒体は、複数種類を利用することが可能で、種類ごとにファイルプールの形式で保持している。記憶媒体となるファイルが不足すると、Schedulerにその割り当てを依頼する。キャッシュ記憶媒体へのアクセスは、通常のVFSインタフェースを利用するので、どのようなファイルシステ

ムでもキャッシュ記憶媒体として利用することができる。したがって、たとえば、Scheduler の実装によっては、read が主体となるキャッシュには DVD を用い、write が主体となるキャッシュには主記憶を利用することも可能である。

5.3 Scheduler の実装

Scheduler は、フレームワーク部分と、それを利用した本体部分からなる。ここでは、フレームワーク部分の実装の概要について述べる。フレームワークは、キャッシュシステム全体を表すオブジェクトと、ファイル個々のキャッシュを表すオブジェクトからなり、DPS が収集した情報にもとづいて、それぞれのインスタンス変数が自動的に更新される。したがって、個々のキャッシュに関する統計情報は、それを表現するオブジェクトのインスタンス変数を参照するだけで、容易に取得することができる。また、このオブジェクトのメソッドを呼び出すことによって、ファイルごとのキャッシュ戦略を変更することができる。

フレームワークは、約 1500 行のソースコードからなり、DPS との通信をオブジェクトの状態変化に変換する機能を持つ。実際のスケジューリング機構は、フレームワークが提供するキャッシュのオブジェクトを継承することによって実装される。

5.4 実証実験

Ver. 1 の開発にあわせて、実証実験の準備をすすめている。大学キャンパス内に複数のキャッシュノードを設置し、Diskless クライアントノードを運用するとともに、専門学校等、複数の組織間での共同運営の準備を進めている。また、家庭にもルータ機能を持つキャッシュノードを設置し、在宅勤務にも適用できるようにする。

このような、Community Storage をもちいた分散協業環境の利点として、

1) 個々の利用者が、ソフトウェアのメンテナンス作業を行う必要が一切ないこと
2) マルチメディア情報も含めて、すべての情報が実時間で共有でき、アップロード/ダウンロードの必要がないこと

等がある。これによって、従来の、Web を主体とした協業基盤よりはるかに使い勝手が向上し、利用者間の緊密な連携が、少ない運用コストで実現できるようになる。

利用分野として、具体的には、eclipse を用いた Java アプリケーションプログラム開発への適用を検討している。Diskless 構成とすることによって、利用者は利用するソフトウェアのインストール、メンテナンスを行う必要が一切なくなる。利用者は、キャッシュ機能付きのルータを設置するだけで利用を開始することができる。

在宅勤務に適した、マルチメディアを利用することのできるグループウェアを開発し、eclipse のプラグインとする予定である。このグループウェアも Community Storage 上で動作する。これによって、たとえば、地域社会において分散環境でプロジェクトを実施する環境が構築できる。

6. あとがき

Diskless クライアントやマルチメディア共有に適用することのできる、大規模分散ファイルシステムの構築方法について提案した。今後、実証実験を重ねた後に、種々の観点からの評価結果について報告したい。

参考文献

1) B. Callaghan, D. Robinson, R. Thurlow, Sun Microsystems, Inc., C. Beame, Hummingbird Ltd., M. Eisler, D. Noveck and Network Appliance, Inc., "RFC3530", http://www.rfc.net/

2) Ralph O. Weber, "Object-Based Storage Device Commands(OSD)", http://www.t10.org-/ftp/t10/drafts/osd/osd-r10.pdf

3) Research group of M. Satyanarayanan, "Coda Papers", http://www-2.cs.cmu.edu/afs/-cs/project/coda/Web/docs-coda.html

4) Linux-iSCSI Project, "driver and daemon for using iSCSI on Linux", http://linux-iscsi.-sourceforge.net/