大規模分散ファイルシステムによる分散コンピューティング環境の構築

小柳順裕[†] 田胡和哉[†] 村田修一郎[‡]
山下直人[†] 兵頭和樹[†] 松下 温[†]

企業、公共組織、学校等、大規模組織において利用される、次世代の分散コンピューティング環境を開発したので、その概要について報告する。ここでは、組織のすべての構成員が利用できる大規模な分散ファイルシステムをインフラとして利用する。このインフラを、Community Storage と名づけた。これを用いて、マルチメディアを含むすべての情報共有、ディスクレス PC の運用等を行うことにより、仮想オフィスの構築、ソフトウェアの集中管理機構が実現できる。現在（2005 年 9 月）、基本機能を全て備えた Community Storage Ver.1 の負荷テストを行っている。またこれと平行して、実証実験の準備を進めている。

Distributed Computing Environment with Large-scale Distributed File System

MASAHIRO KOYANAGI,[†] KAZUYA TAGO,[†] SHUICHIRO MURATA,[‡]
NAOTO YAMASHITA,[†] KAZUKI HYODO[†] and YUTAKA MATSUSHITA[†]

This paper reports the overview of next generation's distributed computing environment that is used in large-scale organizations such as enterprises, public organizations, and the schools. In the proposed method, a large-scale distributed file system that all constituent members of the organization can use is used as an infrastructure. This system was named Community Storage. The construction of virtual offices and the centralized control mechanism of software can be achieved by operating all information including the multimedia sharing and Diskless PC on the Community Storage. The load test of Community Storage Ver.1 that is the basic functions version is running now (September, 2005). Also, we are preparing the demonstration experiment.

本研究は、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。

1. はじめに

企業、公共組織、学校等、大規模組織において利用される、次世代の分散コンピューティング環境を開発したので、本稿においてその概要について報告する。ここでは、ネットワークを利用する組織のすべての構成員が利用できる大規模な分散ファイルシステムをインフラとして利用する。これを、マルチメディアを含むすべての情報、および、ソフトウェアの共有機構として利用することにより、ソフトウェアの集中管理機構、および、時間や場所の制約のない仮想オフィスの構築が実現できる。このシステムの利用者は、オフィスや自宅等、どのような場所においても、均一なコンピュータ環境が提供され、かつ、そのメンテナンス作業を行う必要がなくなる。また、マルチメディアを多用したグループウェアにより、遠隔の地において異なる時間サイクルで作業する人とも共同作業を容易に行えることが期待できる。このような分散コンピューティング環境は、たとえば、在宅勤務環境や協力会社間の共同作業環境の実現のために適用することが考えられる。

大規模な分散ファイルシステムを構築する方法として、集中的なファイルサーバに加えて、ファイルサーバ上のデータのコピーを一時保管するための機能を提供する、キャッシュノードをネットワークの各所に

† 東京工科大学
Tokyo University of Technology
‡ 株式会社アクタスソフトウェア
Acutus Software, Inc.

分散して配置する方式をとる。キャッシュノードは、部門サーバやローエンドルータの付加機能として実現される。キャッシュノードに接続されたクライアントノードは、遠隔ファイルのアクセスプロトコルとして広く利用されている NFS を用いることにより、ファイルを共有することができる。キャッシュノードの働きにより、ネットワークの遅延やスループットの制約が緩和される。

キャッシュノードのこのような機能を利用し、単にファイルを共有するのではなく、OS を含むソフトウェアを分散環境で共有する。これにより、たとえば、ハードディスクを持たない、ディスクレス PC を大規模に利用することが可能になる。これは、キャッシュノード上で PXE サーバを動作させ、PC をネットワークブートさせることにより実現する。ブートするカーネルの選択は、PXE の起動メニューで行う。ディスクレス PC は、それが利用するすべてのソフトウェアを分散ファイルシステム中で共有し、組織中の一箇所でそれらのメンテナンスを行うことを可能とする。これは、TCO の削減にきわめて有効である。また、ソフトウェアは頻繁に変更されることはないので、キャッシュノードが有効に作用する。たとえば、低コストでオフィスのクライアント PC 環境を構築する場合、一般的な性能のハードディスクを各 PC に搭載させるより、キャッシュノードの記録媒体として同じコストの大容量のメモリを用いた方が、クライアント PC の起動時間等の短縮が図れることが期待できる。ディスクレス PC は、デスクトップ環境を全て分散ファイルシステムに保存するので、キャッシュノードを自宅におくことで、オフィスと同じ環境での在宅勤務を実現することができる。

キャッシュノードは、マルチメディアデータの共有も行えるように設計されている。この機能を利用することにより、動画像ファイルのような、生のコンテンツファイルのみならず、たとえば Mpeg7 [1] 形式によるメタデータを含んだファイルを大規模に共有することが可能になり、分散コンテンツデータベースを容易に構築できるようになる。このデータベースは、たとえば、OA 機器やプレゼンテーション記録のオンライン化等を通じて、オフィスにおけるワークフローをオンライン化するために有効である。これを実現し、全ての情報が電子化されれば、在宅勤務や、遠隔地との共同作業も容易に行えるようになると期待される。

以下では、ここで構築する大規模分散ファイルシステムを、Community Storage とよぶことにする。

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、NFSv4 [2] の性能目標と照らし合わせ、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

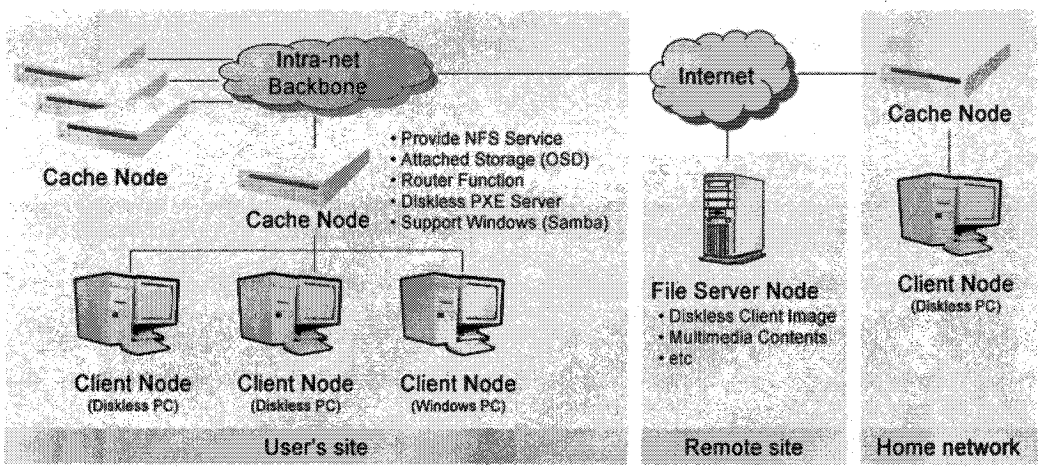

図1 システムの全体構造

て、10000台を超えるクライアントノードを同時に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。認証機構、ユーザ権限の制御等はNFSで利用できるものをそのまま用いる。NFSv4はそもそも大規模な運用を想定しているので、Community Storageにも適用できる。また、キャッシュ動作の対象となるファイルシステムを利用してSambaシステムや、Webプロキシ等をキャッシュノード上で動作させることにより、ファイル分散共有機構を、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NFSv4プロトコル[2]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。また、ファイルサーバノードは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ファイルサーバノードにライトバックされる。ファイル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、集中的にファイルバックアップを行うことを想定する。

キャッシュノードは、

1) ルータとして動作するもの
2) NASとしてプライベートネットワークにアタッチするもの

を検討している。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たルータ装置や、小型のNAS装置内にキャッシュ機能を組み込むことも検討している。キャッシュノードには、ストレージが付加され、自宅に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。

3. 提案方式の特徴

ここで、提案方式の特徴について、既存の方式と比較することによって検討してみる。提案方式の特徴は、マルチメディアの配信機構を含め、すべての情報共有を単一の機構に依存していることにある。これに対して、現状では、情報共有の形態に応じて種々の方式が併用されている。たとえば、ある組織体において、Webコンテンツを共有するためには、必要に応じて複数のサーバを運用すると同時に、ネットワークの状況にあわせて専用のキャッシュを配置することが行われている。現状では、このようなキャッシュはクライアントがコンテンツを取得する(ダウンロード)のために用いられており、他の目的、たとえば、クライアントから情報を発信する目的には用いられていない。また、動画像等のストリーミングデータを配信するためには、別個の配信、キャッシュ機構が用いられるのが通常である。このような方法は、既存のソフトウェアをそのまま利用できる利点がある一方において、いくつかの問題を含んでいる。

そのような問題点の一つとして、ネットワークの利用状況を統括的に把握することが難しくなる点があげられる。複数の情報配信系がそれぞれ連携しながら別個のキャッシュ管理やトラフィック管理を行っているのが現状であり、ネットワークの管理運営をきわめて複雑なものにしている。また、近い将来、このような管理運営の自動化が進むことが予想されるが、その実現を難しくしている。たとえば、オートノミックコンピューティングでは、システムが自身の状況を監視しながら自動的にリソース配分の最適化を行うことをめざしている。情報共有を複数の機構ではなく、一つに集約して実現することは、このような技術の導入において有効に作用する。

今ひとつの問題点として、新たな機能を持つアプリケーションの構築が複雑になりやすい点があげられる。たとえば、ストリーミングデータを扱うワークフロー管理システムやデータベース管理システムでは、メタデータの共有、更新と、コンテンツデータの共有、更新に別個のインフラを利用する必要が生じ、アプリケーションの構築をより難しいものにしている。また、将来新たなフォーマットのコンテンツが出現した場合の対応も難しくなる。

以上の2点の理由により、情報共有機構の集約化を図ったのが提案方式である。ネットワーク管理の容易化や、新たな分散アプリケーションの導入に利点が期待できる一方において、高度なスケジューリング技術が必要となり、実現可能性は自明ではない。ここでは、Community Storageのアーキテクチャの設計上、いくつかの工夫をすることによって、このような目的を達成することを目指している。また、ファイルシステム自体を作ることが目的ではないため、既存のNFS

ファイルシステムを最大限有効に利用している。

4. 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) ファイル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。
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。DPSは、キャッシュ動作中にファイルごとのキャッシュ記憶媒体を変更する機能、ファイルサーバノードへのライトバックの速度を調節する機能、ファイル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。

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の戦略を決定する。
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。

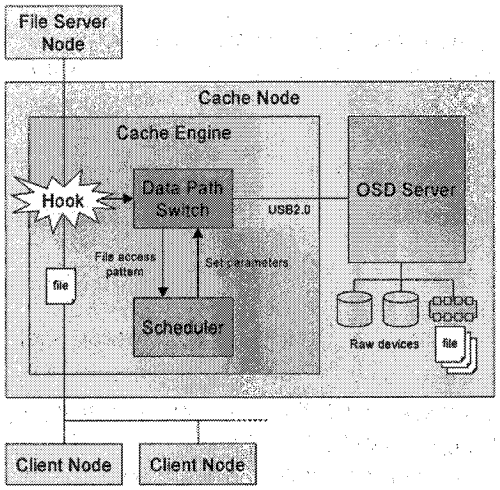

図2 キャッシュノードの内部構造

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内で実行し、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。

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、ファイルごとのパラメータ、キャッシュ記憶媒体を実行時に変更する。

ファイルキャッシュに用いる記憶媒体として、メモリ(RAMファイルシステム)や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[4]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[5]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。

ネットワークプロセッサを用いた、USBインタフェースを備えるルータ装置や、IDEハードディスクやUSBストレージをNASとしてネットワークに接続する装置が市販されている。これらはLinuxで動作するものが多いので、キャッシュノードとして比較的容易に利用できることが期待され、この環境での実装も検討している。しかしながら、このような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大容量の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。具体的には、OSDプロトコルでのファイル読み込みでは、ファイルが属するグループ識別子、ファイル識別子、読み込みを開始するオフセット値、読み込み長等を含んだリクエストを一度やり取りするだけである。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。

5. 現状

5.1 実装

原稿執筆時(2005年9月)に、基本機能を全て備えたVer.1が完成し、負荷テストを行っている。これはカーネルモジュール部分(DPS)とSchedulerのオブジェクトフレームワークの全機能が実装されている。Scheduler本体は単純なアルゴリズムで構成され、自動的な性能チューニング機能は実装されていない。Ver.1のキャッシュノードは、Fedora Core 4、(Linux 2.6.11)を用いて構築されている。現在、Ver.1を用いることによりFedora Core 3によるディスクレスPCが立ち上がることを確認した。また、ディスクレスPC上で、eclipseやOpenOfficeの動作、カーネルビルドが成功することを確認した。

キャッシュノードの実現に加えて、OSDによるキャッシュ記憶システム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によるキャッシュ記憶システムは、キャッシュノード上で動作するカーネルモジュールと、OSD装置上で動作し、OSDプロトコルを解釈実行するエンジンから構成される。OSD装置自体もLinuxで実現され、エンジンはユーザプログラムとして実装されている。OSD装置の記憶媒体には、通常のハードディスクをRAWデバイスとして使用している。現在、基本機能に関して実装が完了している。

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。

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る。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 の完成に合わせて、実証実験が開始できるように準備を進めている。大学キャンパス内に複数のキャッシュノードを設置し、ディスクレス PC 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、自宅にも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。

現在、専門学校と共同で、メタデータ付きのマルチメディアを自由に扱えるようにする、分散マルチメディアメタデータデータベース、および、それを利用した、マルチメディアを自由に扱えるグループウェアを開発している。この開発における分散協業環境として、Community Storage を適用する。現在は個々の PC を用いて開発を進めているが、今後、ディスクレス PC を用いて Java の開発、ソースコードの共有等を行う。移行にあたっては、専門学校にキャッシュノードを設置するだけで利用が開始でき、ソフトウェアメンテナンス、および、ファイルバックアップは、ファイルサーバノード一箇所で行う。また、同様の実験を中国の大学とも行う予定である。

遠隔環境において意思疎通を図るためのツールとして、音声付のデスクトップ画面を動画として記録したり、リアルタイム配信を行ったりするツールを開発している。これは画面記録と同時に、パワーポイントのページめくりイベントや、音声情報をテキスト化したものを取得し、時間軸でそれらを関連付けることで、たとえば、テキスト検索可能な、パワーポイント資料の説明コンテンツを自動作成することができる。

6. あとがき

マルチメディアを含むすべての情報共有、ディスクレス PC の運用等を行うことにより、仮想オフィスの構築、ソフトウェアの集中管理機構が実現できる、次世代の分散コンピューティング環境の概要について報告した。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。また、Scheduler を中心とした自動的な性能チューニング機能に関しても、方式の検討、実装を行い、評価結果を報告したい。

参考文献

1) José M. Martínez, "MPEG-7 Overview (version 10)", http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm

2) 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/

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

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

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