2006－OS－101（11）
2006／2／17

プラグイン方式によるファイルキャッシュ記憶向けリソーススケジューラの実現

小柳順裕[†]　田胡和哉[†]　山下直人[†]

兵頭和樹[†]　松下　温[†]

NFS ファイルシステムに、ファイルキャッシュ機能を提供するノードを追加することで、大規模運用が可能な分散ファイルシステムを開発した。このシステムは多様な環境下で運用されることを想定しており、キャッシュ記憶媒体やネットワーク等のシステム資源のスケジューリングに関して、事前に最適な形式で実現することは困難である。このような観点から、スケジューリング機構のポリシーを分解、階層化することにより、自律的な性能改善機構や、運用後にスケジューリング方針を容易に変更できる機構を構築する。また、I/O バッファを仮想記憶上に取ることによって、キャッシュに用いるハードディスクの Spatial locality を改善する方法を提案する。

Design and Implementation of a Resource Scheduler for File Caching with Plug-in System

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

KAZUKI HYODO[†] and YUTAKA MATSUSHITA[†]

We developed a large-scale distributed file system by adding the nodes that provide the file caching function to the NFS. Since this system is employed in various environments, it is difficult to realize in the optimal form in advance about the scheduling of system resources, such as storage mediums, network, etc. From such a viewpoint, we build an autonomous performance improvement mechanism and the mechanism in which a scheduling plan can be easily changed after starting operation, by making the policy of a resource scheduler into a hierarchic structure. Moreover, we propose the method of improving the spatial locality of a hard disk for file cache, by taking an I/O buffer on virtual memory.

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。

† 東京工科大学
Tokyo University of Technology

1. はじめに

ネットワークシステムの普及にともない、ネットワークを用いて種々のサービスを提供する機構の重要性が高まっている。そこでのネットワーク利用形態として､会議システムのような実時間通信を伴うものと、情報共有、検索のようなデータ蓄積をともなうものに大別できる。ここでは、特に、オフィスや教育機関を対象としたネットワークサービス提供機構に焦点をあて、データ蓄積をともなう大規模な情報共有機構を構築する方式について検討する。具体的には、大規模な分散ファイルシステムをインフラとして利用し、二次記憶装置を持たない、いわゆるネットワークブート型のディスクレス PC を運用したり、分散コンテンツリポジトリを運用したりできる機構を構築する。

この分散ファイルシステムをはじめとする、大規模な情報共有機構では、ネットワークトラフィックを削減し、良好なデータアクセス性能を得るために、データキャッシュ機構が不可欠になる。ハードディスクを用いて、マルチメディアデータやディスクレス PC のためのシステムファイルをキャッシュする機能を持つデータキャッシュ機構を実現した。本稿では、その性能をさらに改善するためのスケジューリング技法について提案する。

実現したシステムは、分散ファイルシステムとして広く利用されているNFS（Network File System）に、データキャッシュを行うための専用ノードを追加することによって構成されている。このキャッシュノードは、NFSプロトコルによるファイルのインポートと再エクスポートを実行するキャッシュ本体と、OSD規格[1]によって接続された外部ストレージから構成される。この分散ファイルシステムを、Community Storageと名づける。

Community Storage は、全体として、多様なアクセスパターンを持つ多数のファイルを保持しており、それが運用されるネットワーク環境も多様である。したがって、Community Storage が利用するネットワーク、プロセッサ、メモリ、ハードディスク等のシステム資源のスケジューリングに関して、事前に最適な形式で実現することは困難である。このため、自律的な性能改善機構や、運用後にスケジューリング方針を容易に変更できるようにする機構の実現が不可欠である。このような観点から、本稿では、特に、OSD 規格に基づいて実現された外部ストレージに焦点をあて、そのシステム資源のスケジューリング機構を構造的に実現することを試みる。ここで開発された技術は、

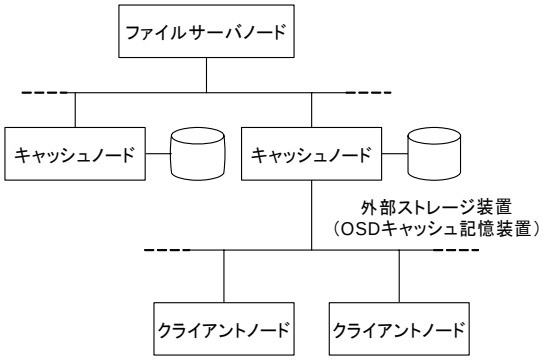

図 1　Community Storage の全体構造

Community Storage の他の資源のスケジューリングにも有効に利用できることが期待される。

システム資源のスケジューリング機構を構造的に実現する一つの方針として、ポリシー/メカニズム[2]の分離が古くから用いられてきた。ここでは、キャッシュのための外部ストレージ機構を対象として、ポリシー/メカニズムの分離を図るだけでなく、複数のポリシーを階層的に連携させるとともに、それをそのままスケジューラのプログラム構造とすることを試みる。これをポリシー階層化とよぶことにする。全体のスケジューリング目標を達成するための機構は複雑になるが、それを複数の部分目標の階層に分解することによって、個々のポリシーが単純化される。また、ポリシー間で必要とされるシステム資源を競合する構造をとり、上位のポリシーにおいて下位のポリシーへのシステム資源分配を決める機能を実現することによって、ポリシーに対応するプログラムを系統的に実現することが可能になる。

ここでは、さらに、キャッシュに用いられるハードディスクのアクセス性能最適化方式として、仮想バッファ方式を提案する。提案方式では、ハードディスクの領域を複数のゾーンとバッファ用の空き領域に分割し、同時にアクセスされる可能性の高いファイルを同一のゾーンに配置する。さらに、アクセスが頻繁にあるゾーンと、隣接するバッファ用の空き領域を仮想記憶機構のページスワップのためのストレージとして利用し、この仮想記憶領域をディスク I/O のバッファ領域としても利用する。一時にひとつのゾーンだけを利用し、他のゾーンに対するアクセスの際にはバッファ領域を経由することによって、ディスクに対するファイル配置の Spatial locality がアダプティブに改善され、単位時間あたりの総シーク距離を削減できる。

以下では、Community Storage の設計、そのキ

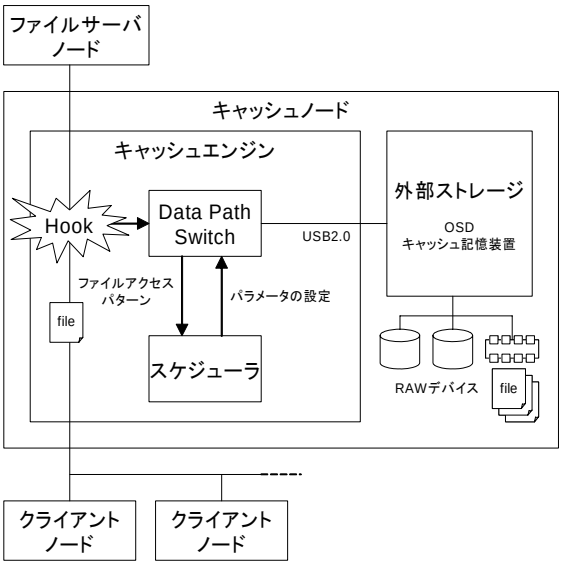

図2　キャッシュノードの内部構造

ャッシュに用いる外部ストレージの設計、スケジューリング方針、仮想バッファ方式、実現状況について述べるとともに、Community Storage システム全体の利用分野についても詳述する。

2. Community Storage システムの構成

図1に、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。

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、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）、および、ストレージ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。Scheduler ではキャッシュノード間のネットワークスケジューリングを行う。

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) VFS の動作を変更することによって、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
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、NFSv4 プロトコル[3]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。

キャッシュノードには、ストレージが付加される。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 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。

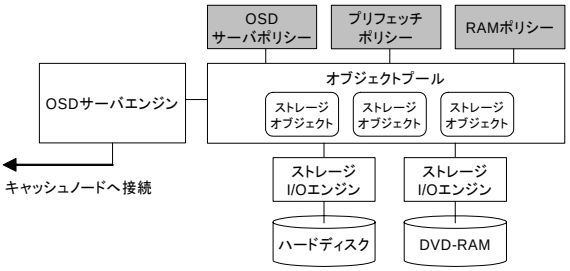

図３　OSD キャッシュ記憶システムの内部構造

このような外部ストレージ装置を、OSD キャッシュ記憶装置と名づける。

3. OSD キャッシュ記憶装置

図 3 に、OSD キャッシュ記憶装置の内部構造を示す。OSD キャッシュ記憶装置は、Linux で実現されており、これらはユーザモードで動作するアプリケーションとして構築されている。キャッシュノードと OSD キャッシュ記憶装置は USB2.0 インタフェースを用いて接続する。OSD キャッシュ記憶装置には、ハードディスクや DVD-RAM、USB ストレージ等の複数の記憶装置が接続され、記憶媒体として用いる。これらの使い分けは OSD キャッシュ記憶装置が行い、キャッシュノードからは、1 つの外部ストレージとして利用される。

OSD キャッシュ装置の設計にあたっては、ポリシーとメカニズムの分離を図るとともに、それをプログラムの構造に反映させることに主眼がおかれている。これに対応して、図３に示すように、プログラムは、

1) ストレージオブジェクト（以下、オブジェクト）
2) ポリシー
3) エンジン

の３つの主要要素から構成される。

オブジェクトはファイルに相当し、オブジェクト ID、オブジェクトサイズ、I/O に関するリソーススケジューリングを行うポリシー(プリフェッチポリシー、RAM ポリシー）へのポインタ、保管先の記憶媒体、アクセスログ等の情報をプロパティとして持つ。これはオブジェクトプールに蓄えられ、キャッシュノードでキャッシュされるファイルに１対１に対応する。

OSD ストレージの資源スケジューリングは、ポリシーがオブジェクトのプロパティを適切に設定することによって達成される。また、プロパティの変更に対応して記憶媒体間でのデータ移動や I/O を実行する、メカニズムに対応する部分をエンジンとよぶ。

OSD サーバエンジンは、

1) キャッシュノードからの I/O 要求に応答する
2) オブジェクトの生成、削除に関する管理を行う
3) キャッシュノードに対する I/O スループットを測定する
4) オブジェクトに対するアクセスパターンのログを収集する

機能を持つ。キャッシュノードからの、Read/Write 等の一部の I/O 要求に対しては、オブジェクトに関連付けられているポリシーへ、要求をディスパッチする。

OSD サーバポリシー、プリフェッチポリシー、RAM ポリシーは、OSD キャッシュ記憶装置内のリソーススケジューリング機構として動作する。各々は別々のスケジューリング目標を持つ。これらは、OSD サーバエンジンから I/O 要求がディスパッチされてきたときや、システム監視時に、適時、オブジェクトに対して命令を出したりパラメータを変更したりする。例えば、オブジェクトにメモリを割り当てて、プリフェッチやファイルキャッシングを行うように命令したり、オブジェクトを保管する記憶媒体を変更したりする。

ストレージ I/O エンジンは、オブジェクトの記憶管理機構を実現する。ハードディスク等の記憶媒体を RAW デバイスとして利用し、独自のファイルシステムを構築する。ストレージ I/O エンジンは、オブジェクトのパラメータにしたがって I/O 処理を行う。

4. OSD キャッシュ記憶装置の資源スケジューリング

4.1 スケジューリング目標

OSD キャッシュ記憶装置のスケジューリング目標は、装置全体としてのスループットを最大化することにある。その際のもっとも重要な点は、ディスク装置の I/O スループット向上を図るための方策である。一般に、ハードディスク装置のスループットは、サーバシステムのボトルネックとなりやすい。これはハードディスク装置が機械的に構成されているためであり、他のすべての構成部分が電子的に構成されているのと対照的である。例えば、1 つのハードディスクに対して、同時に複数の I/O 要求が発生した場合、シークが頻繁に発生し、実時間あたりのデータ転送量は大幅に低下し、ディスク装置の動作時間の大半がシークの実行と回転待ち時間によって占められることが起こる。

以下では、実時間あたりのデータ転送量を実効 I/O スループットとよぶことにする。

OSD キャッシュ記憶装置全体のスループットを向上させるためには、

1) メモリ等、他の記憶媒体をキャッシュとして用いることによってディスク I/O の回数自体を削減する
2) 先読み等の I/O 方式の改善によって I/O 時間に含まれるシーク時間の比率（以下シークコスト）を削減する
3) ディスク装置上でのファイル配置パターンを改善してシークコストを削減する

等の方法が考えられる。

OSD キャッシュ記憶装置内部のメモリを用いてディスク装置のキャッシングを行うことは、負荷のパターンによっては有効であるが、たとえば、マルチメディアデータのような、大きなサイズのファイルに対しては有効でない。

一方、ファイル先読みは、大きなサイズのファイルに対して有効な方法である。すなわち、なるべくディスク上の連続した領域を一時に I/O 処理することにすれば、シークコストが削減される。

このとき、両者ともにメモリを必要とするために、資源の競合がある。したがって、キャッシュ機構と先読み機構のどちらにメモリ資源を割り当てた方がより有効であるかは、負荷の状況によってアダプティブに決める必要がある。キャッシュ機構の評価値は、キャッシュヒット率であり、これによってどれだけディスク I/O 率が削減できるかが決まる。一方、ファイル先読みでは、ディスク装置の実効スループットが評価値になる。それぞれ異なる評価値から、システム全体の目標値への寄与を推測して内部の資源割り当てを決定できるようにする必要がある。

4.2 ポリシー階層化

ポリシー階層、全体のスケジューリング目標を達成するための機構を、複数の部分目標の階層に分解することで、個々のポリシーを単純化する。また、上位ポリシーは、自分の下位に属する複数のポリシーへシステム資源を分配する機能を持つ。上位ポリシーには OSD サーバポリシー、下位ポリシーにはプリフェッチポリシーと RAM ポリシーがそれぞれ対応する。ポリシー間でやりとりされるシステム資源にはメモリが対応する。ポリシー階層化フレームワークを構築し、それを OSD キャッシュ記憶装置に適用する。この様子を図 4 に示す。

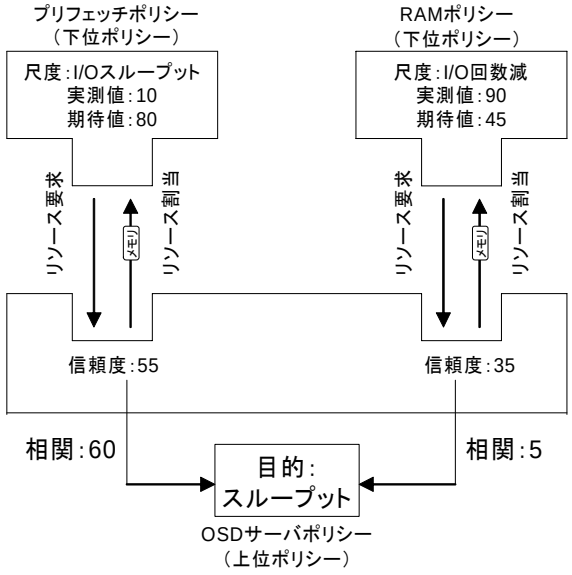

図4　OSD キャッシュ記憶装置のポリシー階層化構造

ポリシー階層化フレームワークの動きは、次のようになる。下位ポリシーは、スケジューリング目標(尺度)、部分目標のスケジューリングに必要なシステム資源の種類と量、現在の性能実測値、要求通りにシステム資源が割り当てられた場合の性能予想値を自己申告する。上位ポリシーも自らのスケジューリング目標を持っており、その性能実測値と、下位ポリシーへのシステム資源配分量の相関を観測する。この経過において、上位ポリシーは下位ポリシーの自己申告する性能予想値の信頼度も決定する。上位ポリシーは、下位ポリシーとの相関、信頼度を見ながら、自らのスケジューリング目標の性能実測値が最大となるように、システム資源を下位ポリシーに分配する。

ポリシーを明確に分割し、かつ、運用開始後に新しいポリシーを追加できる機構を実現するために、ポリシー階層化フレームワークにおける、上位ポリシーから下位ポリシーへの依存関係は疎なものになっている。下位ポリシーは、ポリシー階層化フレームワークが提供するクラスを継承して実装するが、上位ポリシーは下位ポリシーが具体的に何のスケジューリングを行うのかは知る必要がない。上位ポリシーは、自分のスケジューリング目標に、下位ポリシーが何らかの影響を与えることを期待してシステム資源を適当に割り振る。上位ポリシーは観測によって、自分と下位ポリシーの相関、および、下位ポリシーの信頼度を決定す

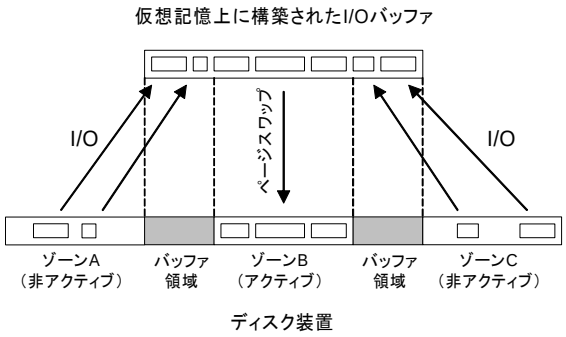

図 5　仮想バッファ方式によるディスク I/O 最適化

る。このとき、ある下位ポリシーがまったく役に立たないか有害、もしくは運用環境に合わずに上位ポリシーのスケジューリング目標の性能向上に寄与しない場合、ポリシー間には相関関係が現れない、もしくは負の相関が現れるので、上位ポリシーはその下位ポリシーを排除することができる。

4.3　仮想バッファ方式

ディスク I/O の最適化を図る今ひとつの方法として、ディスク装置上でのファイル配置パターンの改善、すなわち、Spatial locality の改善があげられる。同時にアクセスされることの多いファイルは、ディスク装置の記憶媒体上でも近い位置に配置されている必要がある。たとえば、最近では、FS2 システムにおいて、ディスク装置上の空き領域にファイルイメージをコピーする方式が試みられている。

このとき、同時にアクセスされるファイル間の関係が、偶然に起因するものと、必然的なものがあり、それらを弁別せずにこのような再配置を行うと効果が制約される問題が生ずる。ここでは、

1) ハードディスクの領域を、複数のゾーンとバッファ用の空き領域に分割し、同時にアクセスされる可能性の高いファイルを同一のゾーンへ配置する
2) アクセスが頻繁にあるゾーンと、隣接するバッファ用の空き領域を仮想記憶機構のページスワップのためのストレージとして利用し、この仮想記憶領域をディスク I/O のバッファ領域としても利用する

ことを試みる。一時に 1 つのゾーンだけを利用し、他のゾーンに対するアクセスの際にはバッファ領域を経由することによって、ディスクに対するファイル配置の Spatial locality がアダプティブに改善され、単位時間あたりの総シーク距離を削減できる。この様子を図 5 に示す。

アプリケーションの種類ごとに、ディスク装置上にゾーンが形成されることが期待できる。たとえば、テキスト編集では、ワードプロセッサのバイナリ、ワードプロセッサから利用されるライブラリ、文書データが、短期間でアクセスされることが期待できる。同様に、プログラム作成、ブラウザの動作でも、短期間でアクセスされるファイル群が存在する。これらのアクセスの特徴は、必然的な理由がある。一方、2 台のクライアントノードで、同時にテキスト編集とシステムビルドが別々に実行されている際、テキスト文書とコンパイラが同時に参照されるのは偶然によるものである。そこで、テキスト文書に関連したゾーンに隣接する空き領域に、テキスト文書に関係しない、コンパイラ等のファイルを集めることによって、単位時間内に実行される積算シーク距離を短くすることができる。

4.4　OSD キャッシュ記憶装置の設計

ここで提案した方式を、OSD キャッシュ記憶装置の設計にどのように反映させるか検討する。

プリフェッチポリシーは、先読みによってディスク I/O 時間に含まれるシーク時間の比率を削減することを目的とする。これはポリシー階層化において、下位ポリシーとして扱われる。ここでは、図 4 で示している尺度として、ディスク I/O スループットが対応する。実測値には、I/O 処理の開始から終了までのディスク I/O スループットのみ測定し、I/O 要求が発生していない時間のものは含まない。

RAM ポリシーは、メモリをファイルキャッシュの媒体として用いることによって、ディスク I/O の回数を削減することを目的とする。これは、下位ポリシーとして扱われる。ここでは尺度として、ディスク I/O 削減回数が対応する。実測値では、Read 要求、Write 要求を区別せずにカウントする。

OSD サーバポリシーは、プリフェッチポリシーと RAM ポリシーへメモリリソースを配分することで、キャッシュノードに対する実効 I/O スループットを最大限に引き出すことを目的としている。これは上位ポリシーとして扱われる。

5.　OSD キャッシュ記憶装置の実現状況

ポリシー階層化フレームワークを C++言語を用いて構築した。これは、上位ポリシー、下位ポリシー、システム資源に関して、それぞれベースクラスを提供

する。フレームワークの利用プログラムは、これらのサブクラスを実装することで、ポリシー階層化を実現する。OSD サーバポリシーは、上位ポリシークラスを継承する。プリフェッチポリシー、および、RAM ポリシーは下位ポリシークラスを継承する。システム資源クラスは、ポリシー間で特定量だけ分配したり、回収したりする。そのため、クラスインタフェースには、インスタンスの分割、結合を行うメソッドが定義されている。ここでは、システム資源クラスを継承して、メモリプールクラスを実装した。

OSD キャッシュ記憶装置、および、ポリシー階層化フレームワークは、一部未実装部分があるものの、外部記憶ストレージとして動作するようになっている。キャッシュノードと OSD キャッシュ記憶装置間は USB2.0 で接続することを予定しているが、現状では TCP/IP を用いて接続している。

ポリシー階層化フレームワークに関しては、一部の実装が完了している。先読み機構は現在実装を行っている。RAM ポリシーはほぼ実装が完了している。ポリシー階層化フレームワークの上位ポリシークラスが持つ、下位ポリシーの評価、リソース配分機能は、現状では単純なアルゴリズムで動作しており、この部分の実装作業も進めている。

仮想バッファ方式に関しては、その詳細な方式を検討している段階であり、実装は行っていない。

評価環境として、OSD キャッシュ記憶装置と直接 OSD プロトコルで接続するベンチマークツールを構築した。正規分布や乱数分布を組み合わせて、特定の傾向を持つアクセスパターンを作成し、シミュレーションを行う。

6. Community Storage システムの利用

6.1 システム全体の実現状況

原稿執筆時（2006 年 1 月）に、基本機能を全て備えた Community Storage Ver.1 が完成している。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 環境を構築している。

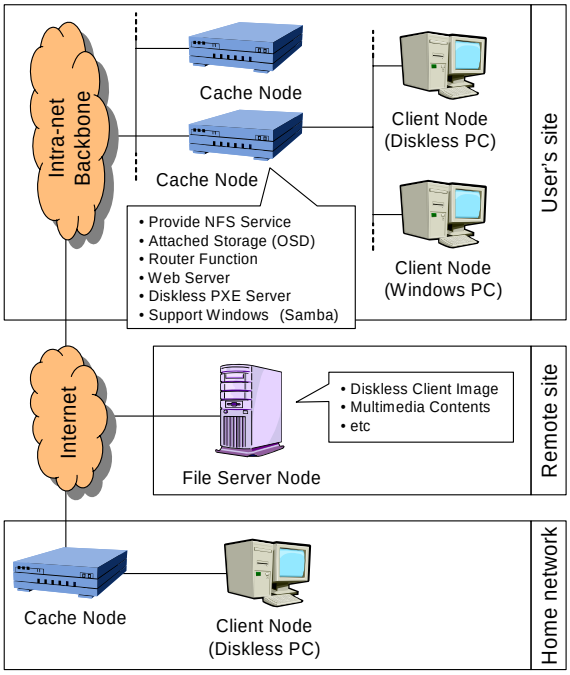

図 6　仮想オフィスシステムの全体構造

6.2 システムの利用

Community Storage は、企業、公共組織、学校等、大規模組織において、その全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用いたネットワークサービスインフラとして利用することを目的としている。これをインフラとした、ディスクレス PC の集中管理環境や、分散マルチメディアリポジトリの開発を行っている。これにより、システム管理コストの軽減、セキュリティの向上、および、マルチメディアを用いたグループウェアシステムをはじめとする新しい分散アプリケーションの実現を図る。

このような環境を構築することの動機の一つとして、労働形態の多様化があげられる。たとえば、政府は 2010 年までに就業人口の 2 割をテレワーカとすることを目標に掲げている。これは就業の一部を自宅等で行うことで実現する。在宅勤務環境は、労働環境の物理的制約の軽減のためのインフラとしての観点から、デジタル社会の重要な構成要素となることが予想される。ここでは、Community Storage を用いて分散仮想オフィスシステムを構築することによって、このようなニーズに応えることをめざす。

図 6 に、ここで実現の対象としている仮想オフィス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。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、

1) 部門サーバ
2) ルータ
3) アプライアンスサーバ

等に組み込まれることを検討している。現状では､PC を用いてキャッシュノードを実現している｡最近では､Linux を用いて実装されているルータやアプライアンスサーバが増えてきている。Linux を用いてキャッシュノードを構築することによって、想定される多くのハードウェアに対応することができる。

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型ディスクレスPCを集中管理するファイルサーバ
2) 分散マルチメディアリポジトリのインフラ

に利用することを想定している。

Community Storage を、ネットワークブート型ディスクレス PC を集中管理するインフラとして用いる。この形式では、PC の動作やアプリケーションの実行に必要な全てのファイルを、オンデマンドにファイルサーバからダウンロードする。これは、アプリケーションの実行をクライアント側で行うので、プロセッサ負荷を分散できること、ネットワークブート可能な多くの既存の PC から利用することができること等の利点を持つ。

ディスクレス PC 運用では、仮想的に、単一のファイルシステムをすべての PC が共有する。したがって、ソフトウェアのアップデートやインストールにおいても、組織全体で一回行えば作業が完了する。このため、システムの管理コスト削減にはきわめて有効であると考えられる。また、クライアント PC が恒常的なデータを持たないために、セキュリティ向上の観点からも有効である。

仮想化されたオフィスでは、業務知識を含む多くの知識は、ドキュメントよりもマルチメディアで蓄積した方が適している。Web 機構を用いて、多数の利用者の間でマルチメディアコンテンツを共有する方法を検討する。従来の、サービスを提供する Web サーバが 1 ヵ所にしかない方法では、マルチメディアによるトラフィックのためにスケーラビリティに制約があり、キャッシュ等の手段が必要になる。ここでは Web サーバを、キャッシュノードに分散して複数配置することにより、Community Storage の機能を利用して Web を用いたマルチメディアアクセスのスケーラビリティを改善する。

構築するシステムでは、共有すべきコンテンツにメタデータを付加して分散リポジトリを構成する。標準インタフェースの 1 つである JCR(Java Content Repository)を用いる。現在、JCR の分散化作業を進めており、複数の Web サーバが単一のコンテンツリポジトリを共有できる構造を構築している。

7. あとがき

自律的な性能改善機構や、運用後にスケジューリング方針を容易に変更できるようにする機構を実現する、リソーススケジューリング機構のポリシー階層化方式について提案した。また、キャッシュに用いられるハードディスクのアクセス性能最適化方式として、仮想バッファ方式を提案した。今後、これらの方式についての性能測定等を重ねた後に、種種の観点からの評価結果について報告したい。

参考文献

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

[2] R. Levin and E. Cohen and W. Corwin and F. Pollack and W. Wuld, POLICY/MECHANISM SEPARATION IN HYDRA, Proceedings of the 5th ACM Symposium on Operating System Principles, 1975

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

[4] Hai Huang and Wanda Hung and Kang G. Shin, FS2: Dynamic Data Replication in Free Disk Space for Improving Disk Performance and Energy Consumption, SOSP'05, 2005