2006－0S－103　(9)
2006／7／31

ファイルキャッシュシステムのストレージの一構築方法

小柳順裕[†]　田胡和哉[†]　山下直人[†]

システムの構成や利用形態が多岐にわたるシステムを、構築当初から最適に運用することは困難であり、運用状況を観測しながらアダプティブに資源割り当て方式を改善し、性能の最適化を図る方法を確立することが急務となっている。本稿では、このような要求に応える一つの方式として、ポリシ階層化方式を提案する。この方式は、複数のスケジューリングアルゴリズムを組み合わせて、系統的に複雑なスケジューラを構築することを可能にする。また、提案方式に基づいてポリシ階層化フレームワークを開発し、これをストレージシステムの構築に適用した。また、このシステムを評価することで、一定のポリシ階層化方式の有効性を確認した。

Construction of a Storage System for File Cache

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

It's not possible to optimize a system that has wide-ranging roles from the beginning of setup. And it's a pressing need to establish the method of performance optimization by observing the operation situation and improving the resource allocation method adaptively. In this paper, we propose the Hierarchical Policy Method. This method enables a complex scheduler to combine two or more scheduling algorithms, and to be constructed systematically. The Policy Hierarchical Framework was developed based on the proposal method, and this was applied to the construction of a storage system. The effectiveness of the Hierarchical Policy Method was confirmed by evaluating this system.

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

計算機システムの分散化、大規模化が著しい。複雑な分散システムにおいて、それを構成する多数の計算機の資源を処理対象に適切に割り当て、処理効率を保つことは容易ではない。特に、システムの構成や利用形態が多岐にわたるために、システムを構築当初から最適に運用することは困難であり、運用状況を観測しながらアダプティブに資源割り当て方式を改善し、性能の最適化を図る方法を確立することが急務となっている。

本稿では、このような、アダプティブに資源割り当てスケジューリングを最適化することを可能にする一つの方法として、ポリシ階層化方式を提案する。そこでの主な狙いは、特定のスケジューリングアルゴリズムを構成することではなく、複数のスケジューリングアルゴリズムを組み合わせて、系統的に複雑なスケジューラを構築することを可能にするソフトウェアアーキテクチャを検討することにある。以下では、スケジューラの内部において、スケジューリング戦略を決める機構をポリシとよぶことにする。異なるスケジュール戦略のそれぞれを個別のポリシとして実現し、同時に運用する。さらに、個々のポリシ（下位ポリシ）の運用実績を評価してそれぞれのポリシに対する資源割り当てを決める、別個のポリシ（上位ポリシ）を階層的に付加することによってスケジューラを構成する。

組み合わされる個々の下位ポリシは、相互に情報を共有せず、互いに独立に構成されているので、たとえば、システム運用後に特定の下位ポリシをよりすぐれたものに交換することや、特定の利用負荷やシステム構成に特化された下位ポリシを追加することが可能になる。また、個々の下位ポリシでは特定のスケジューリング目標を達成すればよいので、下位ポリシを自動的に合成することも可能になることが期待できる。

ここでは、大規模分散ファイルシステム[1)2)]のキャ

† 東京工科大学
Tokyo University of Technology

ッシュとして用いられることを想定したストレージシステムを、提案方式を用いて構築した。このストレージシステムは、ハードディスク、メモリ、プロセッサから構成され、ネットワークを経由して共有されるファイルを一時的に蓄えることによって分散ファイルのアクセス性能を向上させるとともに、ネットワークトラフィックを軽減することを目的としている。この分散ファイルシステムの内容として、通常ファイル以外に、ディスクをもたない、いわゆるディスクレスシステムのシステムファイルや、分散コンテンツ共有機構が保持するマルチメディアファイルが想定されている。したがって、キャッシュストレージシステムの負荷パターンも多岐にわたることが想定される。

下位ポリシとして、ストレージシステムの主記憶をディスクのキャッシュとして利用するものと、先読みバッファとして利用するものの２つを実現し、それらに割り当てる主記憶量を上位ポリシが調節することによって、負荷パターンの変動によっても最適な主記憶割り当てが達成される機構を実現した。ストレージシステムを既存のファイルシステムである EXT2 を用いて実装した場合に比べて、13% 程度平均性能が向上することが観測された。また、負荷パターンが多岐にわたっても、性能の最適化が図れることが確認できた。

本稿では、提案方式、ストレージシステムの構成、評価について述べる。さらに、これを構築した分散システム全体の構成についても述べることにより、提案方式を分散システムの他の資源のスケジューリングについても適用できる可能性について議論する。

2. 方式の提案

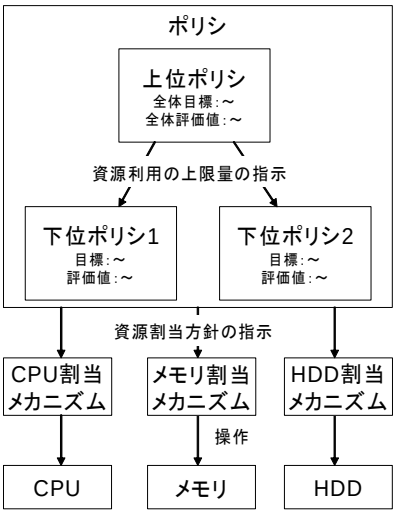

図１　スケジューリング機構の模式図

2.1 方式の概要

図１に、ここで想定するスケジューリング機構の模式図を示す。全体を、メカニズムとポリシに分割して実装することにする[3]。メカニズムは、ネットワーク、プロセッサ、主記憶、二次記憶等の実際の資源を操作する。ポリシは、資源の割り当て方針を決定する。たとえば、プロセッサの割り当てでは、プロセスのコンテクストを切り替える機構がメカニズムに相当し、割り当ての優先度を決めるキューがポリシに相当する。

ここでの議論は、ポリシの構成方式に関するものである。ポリシ機構を、さらに、限定されたスケジューリング目標を達成する複数の下位ポリシに分割して実装する。プロセッサ割り当ての例では、並列数値計算に特化された下位ポリシや、実時間性の高いジョブに特化された下位ポリシを実現することに相当する。それぞれの下位ポリシが、それぞれ固有の評価値を持ち、下位ポリシごとに割り当てられた資源を用いて評価値を最大化するように動作する。

これらの下位ポリシの動作が、全体としてスケジューラ全体の評価値を最大化するように、下位ポリシへの資源割り当て量を決定する、上位ポリシを設ける。すなわち、上位ポリシの評価値はスケジューラ全体の評価値であり、下位ポリシへの資源割り当て量を制御することによって、この評価値を最大化するように動作する。

図２に、上位ポリシと下位ポリシの関係について示す。上位ポリシは定期的に、下位ポリシに必要な資源のリストを提示するように要求する。これは、ボトルネックとなりうる複数資源に関する要求量のリスト

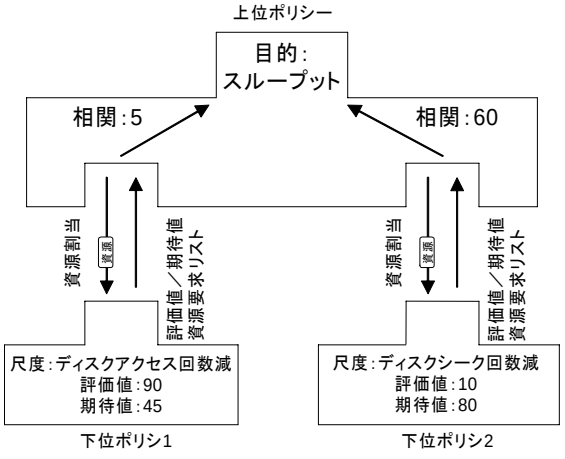

図２　上位ポリシと下位ポリシの関係

である。要求する資源の項目は、同一の上位ポリシの下で動作するすべての下位ポリシで共通である必要がある。上位ポリシは、資源割り当て量を決めた後、同一のデータ表現で資源割り当て量を下位ポリシごとに提示する。実際の資源割り当てを行うのはメカニズムであり、上位ポリシが提示するのはその上限値である。

下位ポリシは、資源の要求を提示するのと同時に、下位ポリシ自身のスケジューリング目標における独自の現在の評価値と、資源の割当が全て満たされた場合に達成することが見込める期待値も提示する。

上位ポリシの評価値と下位ポリシの評価値の関係性は、事前に明らかにされていない。上位ポリシが統計的な手法によって相関関係を解析する。また、下位ポリシの評価値の期待値の推定がどの程度正確であるかも、統計的な手法によって解析する。これらによって、上位ポリシは、下位ポリシ内部の論理に依存せずに下位ポリシに対する資源割り当て量を決めることができる。

このような、上位ポリシと下位ポリシの関係を、スケジューラをプログラムとして実現する立場から考えてみると、おおきな利点であることがわかる。すなわち、これは、上位ポリシが、下位ポリシの内部構造を先験的に知っている必要がないことを意味している。したがって、上位ポリシを実装したあとで下位ポリシを必要に応じて追加したり、内部の論理を改良したりできることになる。たとえば、オブジェクト指向言語を用いてこれらを実装すれば、スケジューリング機構のモジュラリティを高め、下位ポリシをモジュールとして個々独立に実装したり変更したりすることができるようになる。

スケジューリング機構を自動的に構成する試みはこれまでも種々なされてきた。基本的には、スケジューリング機構を外部から観測して動作を評価し、パラメータの変更等の方法によって改善を加える方法がとられている。その際、外部の観測系がスケジューリング機構のセマンティクスをまったく知らないと、条件が複雑すぎて有効な観測が行いにくい。一方において、外部の観測系がスケジューリング機構のセマンティクスに依存して作られている場合には、スケジューリング機構の実装を変更する際に観測系も変更しなければならなくなる。

提案方式は、スケジューリング機構自体が、下位ポリシ、観測系が上位ポリシに相当する。期待値の形式で、自らの評価の手がかりとなる値を申告する方式をとることによって、上位ポリシの実装を下位ポリシの実装とは独立に行えるようにしている。すなわち、下位ポリシが提示する期待値は、現在の負荷状況がその下位ポリシが持つアルゴリズムのどの程度適合するかの度合いを示している。たとえば、プロセッサ割り当ての例では、並列度の高いプログラムが実行されれば並列処理用の下位ポリシの期待値が高くなる。上位ポリシは、並列処理に関する知識がなくても、並列処理用の下位ポリシに適切に資源割り当てを行うことができる。このとき、上位ポリシは下位ポリシが提示する期待値を一方的に信頼するのではなく、期待値自体の妥当性を観測することによって、全体として適切な観測系を構成することができる。

2.2 ストレージシステムへの適用

提案方式は、分散環境における種々のスケジューリング、たとえば、ストレージシステム、ファイルキャッシュシステムのキャッシュ間連携におけるファイル移動のスケジューリングや、ジョブの分散実行におけるプロセッサ割り当てのスケジューリングに適用することを想定している。OSD[4]ストレージシステムを例にとり、提案する方式について説明する。

2.2.1 スケジューリング戦略

OSD ストレージシステムは、ディスク装置以外に、主記憶とプロセッサをそなえている。これらの資源を有効に利用して、ファイルサイズやアクセス頻度に関する、広範な負荷パターンにおいて良好な性能を得るための戦略について考えてみる。

ディスク装置を効率よく運用するためには、シークの頻度をなるべく削減する必要がある。このためには、一度の I/O 単位を大きくとる必要がある。特に、マルチメディアコンテンツファイルのような、大きなサイズのファイルは、I/O バッファのための主記憶割り当てが可能な範囲で、なるべく I/O サイズを大きくとることが有効である。

一方において、アクセス頻度の高いファイルに関しては、当然ながら、主記憶をキャッシュとして用いることが有効である。

両方針は、主記憶の利用に関して競合している。また、異なる特徴を持つ。前者は、大きなファイルのアクセスに際して、その I/O 効率を改善する効果があり、単一のファイルに対する 1 回かぎりのアクセスでも有効である。後者は、I/O 回数自体を削減する効果があり、単一のファイルに対して複数回のアクセスが必要となる。

それぞれの方針を実現する子ポリシを実現し、親ポリシが両者に対する主記憶割り当てを、負荷状況に

あわせて調停する構造をとることにより、全体として多様な負荷に対して最適な性能を実現できる可能性がある。

2.2.2 各ポリシのアルゴリズム

以下に OSD ストレージシステム（以下、特にことわらない限り、システム）を実装するために必要な運用される親ポリシ、および、子ポリシのアルゴリズムアルゴリズムについて述べる。

親ポリシの評価値は、システム全体の評価値とおなじであり、ここでは、アクセス　スループット（byte/sec）である。親ポリシは、システム全体のアクセススループットを最大化することを目的として動作する。

下位ポリシは、

1) データを主記憶にキャッシュする方針を決めるキャッシュポリシ
2) データを先読みする方針を決めるプリフェッチポリシ

の 2 つを実現する。これらの評価値は、それぞれ、

1) キャッシュヒットにより削減できたディスクアクセス量
2) 先読みにより削減できたディスクシーク頻度

から、それぞれ独自の単位を用いて定義される。また、現在の I/O パターンに対して自分の方針がどの程度有効かを各々が判断し、親ポリシに対して、資源の要求と、評価値と同一の単位による期待値の提示を行う。

このとき、上位ポリシのアルゴリズムは、下位ポリシのスケジューリング目標にも、資源の種類にも依存しない。上位ポリシが行うのは、それぞれを区別した上で、評価値や期待値等の数値を比較し、資源割当量を数値で出すのみである。これにより、上位ポリシは再利用可能な、共通化されたアルゴリズムを構成することができる。

3. 実装

提案方式を用いて、OSD ストレージシステムを構築した。提案方式によって得られる、プログラム記述上の利点として、

1) 親ポリシが、スケジューリング対象や子ポリシのアルゴリズムに依存せず、再利用できること
2) 全体の構造が、オブジェクト指向をもちいたフレームワークによく適合すること

が指摘できる。

ここでは、ポリシ階層化方式を、C++言語を用いて、フレームワークを実装した。フレームワーク化によって、以下のメリットが生まれることが期待できる。

1) 上位ポリシや、上下ポリシ連携部分などを、毎回コーディングする必要がなくなる
2) フレームワークで用意されたベースクラスを継承しカスタマイズするだけで、下位ポリシが実装できる。この際、上位ポリシや下位ポリシ間のインタフェースはフレームワークで定義されるので、一々考慮する必要がない

以下では、最初に、OSD ストレージシステムの外部インタフェースの説明をしたあとに、フレームワークの実装を含む、ソフトウェアの実装に関して述べる。

3.1 OSD ストレージシステムの外部インタフェース

OSD のコマンドインタフェースを表 1 に示す。OSD は、記憶装置にファイルシステム機能の一部をストレージシステムにオフロードし、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。OSD では、ファイルに相当するオブジェクトと、それを束ねるグループからなる。オブジェクトに対する I/O（read、write コマンド）は、オブジェクト ID、I/O を開始するオブジェクトオフセット、I/O サイズによって行われる。

OSD は通常、iSCSI 等のストレージネットワークや、ファイバチャネルによって接続された大型のストレージシステムに用いることを想定しているが、ここでは、OSD クライアントと OSD ストレージシステム間は USB2.0 で接続することを予定している。現状では TCP/IP を用いて接続している。

OSD 規格にしたがってストレージシステムを構築する場合には、ファイルシステムのディレクトリ管理機構を除いた部分をストレージシステム内部におい

表 1　OSD コマンドインタフェース

コマンド名	動作
create_group	グループの作成
remove_group	グループの削除
create	オブジェクトの生成
create_group	オブジェクトの削除
write	オブジェクトへのデータ書き込み
read	オブジェクトからのデータ読み込み
set_attribute	オブジェクト属性の設定
get_attribute	オブジェクト属性の取得

て実装する必要がある。一方において、ストレージシステムを単位とした性能最適化のスケジューリングは、従来の、ブロックを単位としたストレージシステムより容易に実装できるようになる。また、複数の記憶媒体を用いたストレージシステムの仮想化も行いやすい。たとえば、ストレージシステム内部においてメモリ、ハードディスク、フラッシュメモリを、ファイルの性質に応じて使いわけることが、従来より的確に行えるようになる。

3.2 ソフトウェアの構造

ここではソフトウェア構造を、大きく、

1) ポリシ階層化フレームワーク
2) フレームワークを用いて実装された OSD ストレージシステムのポリシ群
3) OSD ストレージシステムのメカニズム

の 3 つに分け、それらの実装を順番に述べる

3.2.1 ポリシ階層化フレームワークの実装

ポリシ階層化フレームワークは、主に、

1) ルートリソースクラス（policy::root_resource）
2) リソースクラス（policy::resource）
3) 下位ポリシクラス（policy::lower_policy）
4) 上位ポリシクラス（policy::upper_policy）

から構成される。特に、ルートリソースクラス、リソースクラス、下位ポリシは、純粋仮想関数のみで構成されたインタフェースクラスである。

ルートリソースクラス、および、リソースクラスは、メカニズムとのインタフェースの役割を果たす。各ポリシは、これらを用いて有効資源量を観測したり、利用可能な資源量の上限を設定したりする。ルートリソースクラスは上位ポリシが使用する資源の各々に 1 つずつ実体が割り当てられる。リソースクラスは、個々の下位ポリシが使用する資源の各々について、下位ポリシの各々に実体が割り当てられる。

ルートリソースクラスは、ひとつのシステム資源を集中管理するためのクラスである。例えば、OSD ストレージシステムでは、ルートリソースクラスを派生して、メモリルートリソースとディスクルートリソースが実装されている。今回は実装していないが、この他には CPU ルートリソースや、ネットワークルートリソース等がこれに当たる。インタフェースとして、資源のタイプを表す文字列を返す関数、ルートリソースクラスに関連づいた新しいリソースクラスインスタンスを生成する関数、管理資源の総量、空き資源量などを問い合わせる関数等を持つ。

リソースクラスは、各下位ポリシからのルートリソースクラスに対する限定されたアクセスを実現するインタフェースの役割を持つ。これは、プロパティとして資源の利用可能上限値をインスタンスごとに持ち、関連付けられた下位ポリシは、その上限値の範囲でのみ、資源の利用が可能になる。この派生クラスは、ルートリソースクラスの派生クラスと対応している必要がある。例えば、メモリルートリソースクラスに対して、メモリリソースクラスが存在する。

下位ポリシクラスは、分割されたポリシの 1 つを実装するためのクラスである。具体的なインタフェースとしては、上位ポリシの要求に応じて評価値、期待値、資源要求リストを提示するための関数、上位ポリシが、下位ポリシの資源割当量の変更を通知するための関数等から構成される。下位ポリシの実装時には、これらの上位ポリシ連携のためのいくつかの関数を実装する必要がある。

上位ポリシは分割された下位ポリシを統合し、統計的な手法により資源の割当を行う、ポリシ階層化フレームワークにおいて最も重要なクラスである。また、フレームワークが提供する他のクラスと異なり、すでに実装が完了した、再利用可能なクラスになっている。アプリケーションで行わなければならないのは、上位ポリシ自身の評価値を得るための関数を 1 つ実装するだけである。

上位ポリシは、自身の評価値と、下位ポリシ評価値の相関を分析し、下位ポリシの期待値を参考にしながら競合する資源を配分する機能を持つ。上下ポリシ間の評価値の相関関係はニューラルネットワークを用いて学習される。これは各下位ポリシの評価値を入力とし、上位ポリシの評価値を教師信号として学習が行われる。これは、下位ポリシの評価値の推定値から、上位ポリシの評価値の推定値を導き出す関数として利用している。

上位ポリシは定期的に、下位ポリシの評価値、期待値、要求資源リストを収集する。すると、入力を下位ポリシの評価値、教師信号を自身の評価値として、ニューラルネットワークの逐次学習を行う。また、これらの値は、データベースに保存され、数分に一度の頻度で、学習誤差の減衰率が一定値以下になるか、一定の学習回数を超えるまで、繰り返し学習が行わる。

次に、上位ポリシは、下位ポリシの資源要求リストから競合する資源を探し出し、そこから制約条件を

導き出す。上位ポリシは、これに基づいて、いくつかの資源割当案を策定し、それぞれの案を試行した場合に得られる自身の評価値の推定値を比較することで、最も有効な割当案を選択し、実行する。推定値を算出する方法は次のようになる。最初に、割当案試行時の、各下位ポリシの評価値の推定値を算出する。これは、下位ポリシが要求する資源量を満たすことができる場合は、期待値を推定値とし、満たさない場合は、最もボトルネックとなる資源の要求満足度に比例して、現評価値と期待値から推定値を導き出す。下位ポリシの期待値推定が完了すると、これらの値をニューラルネットワークに入力し、その出力を上位ポリシの評価値の推定値とする。

3.2.2 フレームワークを利用したポリシ群の実装

ここでは、OSD ストレージシステムにおいて実装したポリシに関して記述する。

上位ポリシに関しては、フレームワークにあるベースクラス policy::upper_policy の機能に、上位ポリシの期待値を得るための関数を追加した。これは現在の OSD クライアントに対するスループット（Byte/sec）を返す。

下位ポリシは、以下の 3 つを定義、実装した。

(1) osd::storage_policy
(2) osd::cache_storage_policy
(3) osd::prefetch_storage_policy

これらは、図 3 に示す継承関係を持つ。osd::storage_policy は、OSD ストレージシステムにおける下位ポリシを実装する上で必要な、追加インタフェースが定義されている。具体的には、ストレージオブジェクトへの I/O 要求発生時に、OSD メカニズム側が下位ポリシからの資源利用に関する命令を受け取るためのインタフェースが定義されている。

更にこれを継承し、osd::cache_storage_policy、および、osd::prefetch_storage_policy を実装している。これらはそれぞれ、キャッシュ、先読みに関する具体的な実装が含まれる。

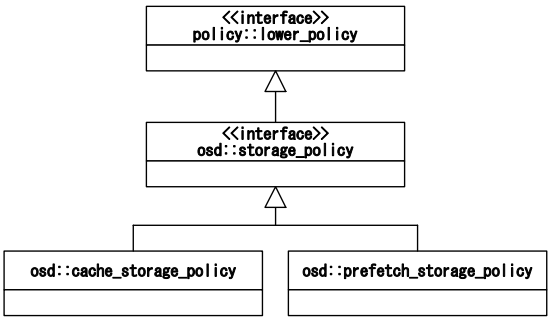

図 3　ポリシ群の継承関係

3.2.3 OSD ストレージシステムのメカニズムの実装

OSD ストレージシステムのメカニズムを構成する主なクラスを以下に示す。

(1) osd::group
(2) osd::object
(3) osd::server_interface
(4) osd::server_engine
(5) osd::storage_engine

また、ポリシ群を含めた、これらの関係を図 4 に示す。

osd::group、および、osd::object は、それぞれ OSD プロトコルで定義されるグループとオブジェクトを表すクラスである。osd::server_interface は、表 1 に示した OSD コマンドインタフェースを定義している。osd::server_engine は osd::server_interface を継承し、OSD プロトコルの解釈、グループとオブジェクトインスタンス群の管理、I/O 要求の下位ポリシへのディスパッチ、および、下位ポリシからの資源割当の方針を osd::storage_engine へ通知する等の役割を持つ。また、上位ポリシの期待値を得るための関数を提供する osd::storage_engine は、ファイルシステムの記憶管理、ページング機構の役割を持ち、下位ポリシから指示された戦略を遂行する部分でもある。

4. 評価

この章では、ポリシ階層化フレームワークを利用して実装された OSD ストレージシステムに人工的な

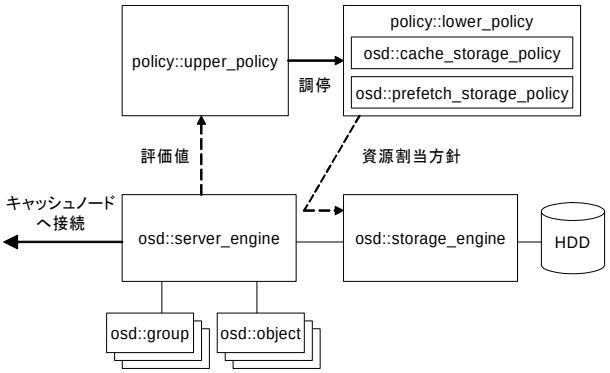

図 4　OSD ストレージシステムの内部構造概略図

負荷をかけることで、ポリシ階層化が適切に動作しているかどうかを確認する。ここでは、以下の 2 種類の評価を行った。

1) ポリシ階層化動作を有効化し、資源をポリシ間で動的に割り当てた場合と、各下位ポリシへの資源割当量をあらかじめ固定した場合のスループットを比較し、資源を動的に配分することの有効性を確認する

2) ポリシ階層化を用いて実装された OSD ストレージシステムの場合と、Ext2 を利用して構築された OSD ストレージシステムの場合のスループットを比較し、ポリシ階層化方式の有効性を確認する

測定は、テストプログラム（OSD クライアント）から、時間の経過とともに変化するパターンで、OSD ストレージシステム（OSD サーバ）へ I/O 要求を発行し、それに対するスループットを測定する方式をとった。

テストプログラムから発行される I/O 要求は次のようなものになる。I/O 要求は 90 秒間、発行し続けられる。このうち最初の 30 秒間は、キャッシュ対象ファイルと先読み対象ファイルへアクセスする比率は 99:1 になる。次の 30 秒間は、この比率が 10:90 に変動する。残りの 30 秒間は、99:1 に戻る。

キャッシュ対象となるファイルと、先読み対象となるファイルの構成を表 2 に示す。また、下位ポリシが利用できるメモリの総量は 50M とする。

なお、OS は Linux Kernel 2.6.12 を用い、テストプログラムと OSD ストレージシステムは同一の PC 上で実行した。

4.1 ポリシ階層化動作時と資源割当量固定時の比較

ここでは、

1) ポリシ階層化を有効にした場合
2) キャッシュポリシに 49MB、プリフェッチポリシに 1MB を固定で割り当てた場合（以下、固定 1）
3) キャッシュポリシに 40MB,プリフェッチポリシに 10MB を固定で割り当てた場合（以下、固定 2）

の 3 種類に関して、スループットを測定し、比較した。

測定結果のグラフを図 5 に示す。横軸は経過時間（秒）を、縦軸はスループット（MB/sec）を表す。また、グラフ中の 2 本の縦の点線は、負荷パターンが変化した時点を表す。左側から、ゾーン 1、ゾーン 2、ゾーン 3 と名づける。ゾーン 1，3 では、サイズの小さいファイルへのアクセスの比率が高く、ゾーン 2 では大きなファイルへのアクセスの比率が高い。

固定 1 のケースでは、ゾーン 1，3 では性能が良好である一方、ゾーン 2 では最も性能が悪くなっている。これは、先読み効果が十分に得られていないことを示している。

逆に、固定 2 のケースでは、ゾーン 2 での性能が良好である一方、ゾーン 1，3 での性能が他の動作条件に比して悪い。これは、キャッシュの効果が不十分

表 2 ファイル構成

	ファイルサイズ	ファイル数
キャッシュ対象	1MB	50
先読み対象	100MB	10

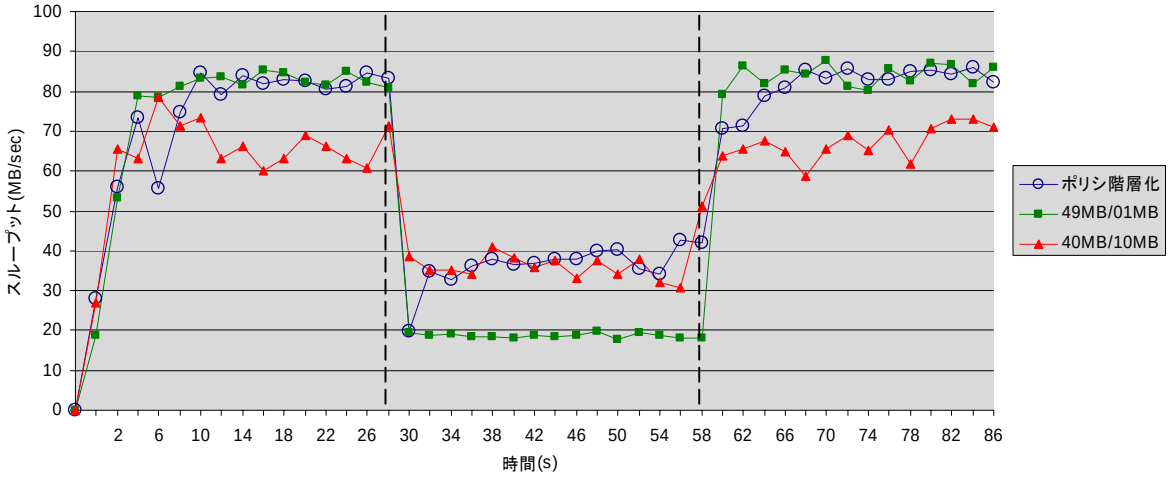

図 5 測定結果 1

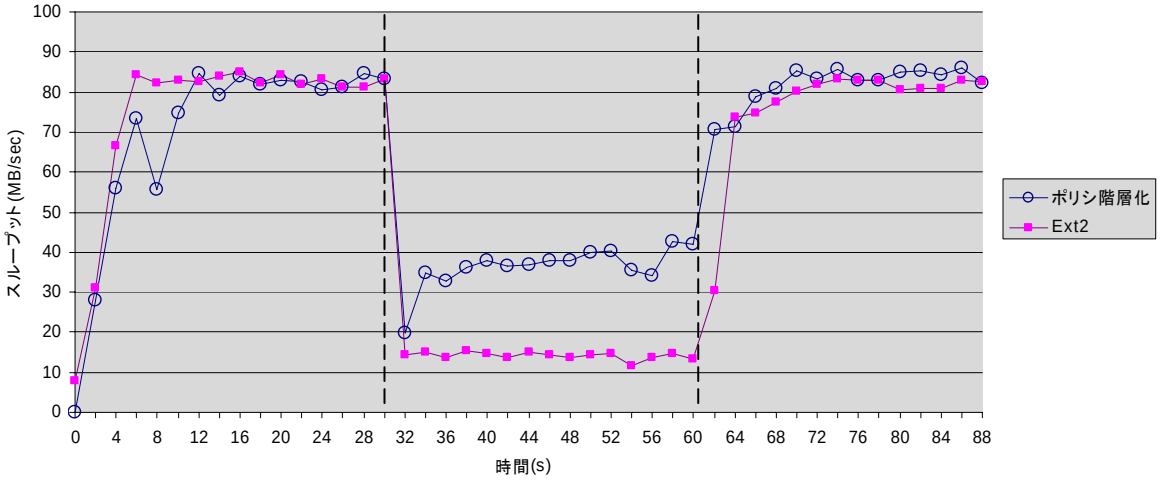

図 6　測定結果 2

表 3 平均スループット値

パターン	平均スループット値(MB/sec)
Ext2	54.9
ポリシ階層化	62.3

であるあることを示している。

提案方式によるシステムでは、双方の条件で、良好な性能が得られる。全体として、提案方式によるシステムが、負荷の状況によってアダプティブに資源割当を最適化し、性能の最適化を図るように動作していることが確認できた。

4.2　ポリシ階層化方式と Ext2 利用時の比較

ここでは、

1) ポリシ階層化を用いた OSD ストレージシステム
2) Ext2 を利用して構築された OSD ストレージシステム

の 2 つのシステムで、4.1 と同じ測定を行い、結果を比較した。

Ext2 による OSD ストレージシステムの OSD プロトコルを解釈する部分は、ポリシ階層化を用いたシステムものと同じ構造を持つ。また、OSD のオブジェクトに通常ファイルを対応付けてデータ記憶管理を行っている。よって、キャッシュ、および、先読みスケジューリング動作は VFS の実装に依存する。

評価結果のグラフを図 6 に、平均スループット値を表 3 に示す。ここから、今回の I/O アクセスパターンにおいては、ポリシ階層化を用いた OSD ストレージシステムは、Ext2 によるそれよりも 13% 程度の性能向上が確認できた。

5.　おわりに

複数のスケジューリングアルゴリズムを組み合わせて、系統的に複雑なスケジューラを構築することを可能にする、ポリシ階層化方式を提案した。また、提案方式に基づいて、ポリシ階層化方式の実装を支援するフレームワークを開発した。これを用いて OSD ストレージシステムを構築し、人工的な I/O アクセスパターンによる測定を行い、一定のポリシ階層化方式の有効性を確認することができた。今後、実システムの運用に基づく評価結果について報告したい。

参考文献

1) 小柳順裕, 田胡和哉, 山下直人, 兵頭和樹, 松下温：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, SWoPP2005
2) 小柳順裕, 田胡和哉, 山下直人, 兵頭和樹, 松下温：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, 第 101 回 OS 研究会
3) R. Levin, E. Cohen, W. Corwin, F. Pollack, W. Wuld : POLICY/MECHANISM SEPARATION IN HYDRA, Proceedings of the 5th ACM Symposium on Operating System Principles, 1975
4) Ralph O. Weber : Object-Based Storage Device Commands (OSD), http://www.t10.org/ftp/t10/drafts/osd/osd-r10.pdf