ポリシ階層化方式によるストレージシステムの実現と評価

小柳順裕[†] 田胡和哉[†]
市村哲[†] 松下温[††]

システムの分散化にともないその構成や利用形態が多岐にわたり，構築当初から最適に運用することが難しくなっている．そこで，運用状況を観測しながら資源割当て方式を改善し，性能の最適化を図る方法の確立が急務となっている．この要求に応える 1 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．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．このシステムの評価を行うことで，ポリシ階層化方式の有効性を確認し，また，EXT2 によるシステムと比べて大きな性能改善が見られた．

Implementation and Evaluation of a Storage System by Policy Hierarchy Method

Masahiro Koyanagi,[†] Kazuya Tago,[†] Satoshi Ichimura[†] and Yutaka Matsushita[†]

It is hard to optimize a system that has wide-ranging roles from the beginning of setup. And it is a pressing need to establish the method of performance optimization by observing the operation situation and improving the resource allocation method adaptively. As a method which meets this demand, we propose the Policy Hierarchy Method. This method enables a complex scheduler to combine two or more scheduling algorithms, and to be constructed systematically. The Policy Hierarchy Framework was developed based on the proposal method, and was applied to the construction of a storage system. The effectiveness of the Policy Hierarchy Method was confirmed by evaluating this system. Moreover, the big performance improvement was observed compared with a system by EXT2.

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段階的に資源割当て方式を改善し，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，段階的に資源割当てアルゴリズムを最適化することを可能にする 1 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．同一の資源に関する異なるスケジューリング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

† 東京工科大学
Tokyo University of Technology
†† 住宅情報化推進協議会
ALICE FORUM

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．

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のキャッシュとして用いられることを想定した OSD ストレージシステム[1] を，提案方式を用いて構築した[2),3)]．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遠隔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に保持されるファイルの内容として，通常ファイル以外に，ディスクを持たない，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予想される．

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当てが維持される機構を実現した．これによって，プログラムのビルド作業とマルチメディアコンテンツアクセスが混在する負荷パターンにおいては，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，大きな性能向上が得られ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．

2. 方式の提案

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[4] に分割して実装することにする．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，資源の割当て方針を決定する．たとえば，プロセッサの割当てでは，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．

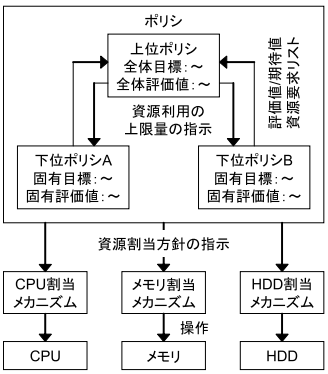

図 1 スケジューリング機構の模式図
Fig. 1 Pattern diagram of a scheduling mechanism.

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当て量を決定する上位ポリシを設ける．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当て量を制御することによって，その評価値を最大化するように動作する．プロセッサ割当てのスケジューリングの例では，複数の下位ポリシへのプロセッサ時間の配分を制御することによって，全体として処理スループットが最大化されるようにする．

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，資源割当て量を決めた後，資源割当て量を下位ポリシごとに提示する．実際の資源割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，下位ポリシ自身の固有のスケジューリング目標に関して，現在の評価値と，資源の割当て要求がすべて満たされた場合に達成することが見込める将来の評価値に関する期待値も提示する．

上位ポリシの評価値と個々の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，大きな利点であることが分かる．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たフレームワークによっ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最適化する試みはこれまでも様々になされてきた．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アルゴリズム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アルゴリズム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．

上記の例において提案方式は，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．下位ポリシが，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，内部のアルゴリズムを知らなくても有効な観測が行えるようにした．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．

以上より，提案方式は，

(1) スケジューリングアルゴリズムを，互いに資源を競合する複数の部分アルゴリズムとその評価機構に分割して実装することによってアルゴリズムの段階的改善を可能にしていること，
(2) 部分アルゴリズムごとに別個の評価単位を設定するとともに，その期待値を自己申告することによって，部分アルゴリズムが作りやすく，かつ，その評価が容易に行えるようにすること，

を特徴としている．

下位ポリシB評価値

下位ポリシA評価値	100	110	120	130	140
1	10	20	30	40	50
2	15	27	38	50	75
3	30	38	59	70	100
4	80	99	128	180	260
5	130	180	200	260	350

図 2 スケジューリング戦略表の例
Fig. 2 An example of a scheduling strategy table.

2.2 上位ポリシの詳細動作

上位ポリシは資源の種類によらず，システム全体の評価値を返す関数を除いて，単一の実装を利用することができる．上位ポリシは，

(1) 各下位ポリシの目標性能を決める
(2) 各下位ポリシの期待値から，目標性能を得るために必要な資源量を計算する

ことを周期的に繰り返している．この繰返しをスケジューリング周期とよぶことにする．

各下位ポリシが発揮すべき目標性能の決定にもちいられる表の概念図を**図 2** に示す．ここでは，A と B の 2 つの下位ポリシが存在することを想定している．この表は，下位ポリシ A，B の評価値からシステム全体の評価値を推定するためのものであり，各項目には，推定されるシステム性能が記されている．これは，学習機構によって作成される．表の縦軸は下位ポリシ A の評価値を特定の粒度で分割し，横軸は下位ポリシ B の評価値に関して分割している．この表をスケジューリング戦略表と名づける．

上位ポリシのアルゴリズムは，ある時点のシステム性能値から出発する．これは，表中の特定の項目に対応する．スケジューリング周期ごとに下位ポリシに対する資源割当て量の変更案を複数作成する．資源割当て量の変化から，下位ポリシごとの期待値を用いることによって，次のスケジューリング周期における下位ポリシごとの評価値が推定できる．したがって，個々の変更案は，図中の矢印で表されるように，スケジューリング戦略表の項目間での移動に対応する．複数の変更案の中から，高いシステム性能が得られるものを選択する．

このように，下位ポリシの期待値を用いることによって，ポリシ間の相互依存関係を考慮することなく，単純な方法で最適な資源割当て案を作成することができるようになる．

2.3 提案方式の目的

提案方式の目的は，スケジューラを，互いに内部の実装に関する情報を共有しない，複数の要素に分割して実装できるようにすることにある．提案方式による最小構成のスケジューラ S1 は，メカニズム M，上位ポリシ U，下位ポリシ L1 の組 S1(M, U, L1) によって構成される．それぞれの要素は他の要素の内部実装に関する情報を用いずに実装されており，要素間の連携の方法のみが，フレームワークによって規定されている．このとき，さらに，S1 の運用を開始した後に別個の下位ポリシ L2 を新たに実装して追加し，スケジューラ S2(M, U, L1, L2) を構成することが可能である．M，U，L1 は，L2 の存在を仮定せずに実装されているが，L2 を追加しても S2 はスケジューラとして動作する．さらに，U は，L1，L2 の特性を観測して，それぞれに資源を配分することにより，動的にシステムの負荷条件が変化しても最適に近い状態でスケジューラを動作させるように機能する．

提案方式は，以下の利点を持つことができる．

(1) プログラム記述の容易化
互いに内部の実装に関する情報を共有しない複数の要素に分割して実装することは，すなわち，プログラムのモジュラリティが，全体を 1 つのプログラムとして実現する従来の方法より改善されることを意味するので，プログラムの実現が容易になるはずである．

(2) 運用状況に合わせた段階的な性能改善の容易化
メカニズムや上位ポリシを先に実装してシステムを運用した後でも，新たなスケジューリングアルゴリズムだけを実装して追加することが可能になることを意味するので，運用状況にあわせた段階的な性能改善が容易になるはずである．スケジューラ全体を 1 つのプログラムとして実装した場合には，新たなスケジューリングアルゴリズムを追加する際に，そのプログラムを実装するばかりでなく，既存のプログラムに新たなプログラムが追加されることによる影響を分析して，必要があれば既存部分に変更を加える必要がある．

提案方式が実際にこのような利点を持つことを確認するためには，提案方式を適用したシステムを構築して検証する必要がある．特に，スケジューラ全体を 1 つのプログラムとして実装し，プログラマが全体として最適に動作することを設計時に確認する作業を行わなくても，種々の負荷パターンや負荷パターンの時間変動に対して全体として最適に近い動作が維持されることを評価によって示す必要がある．また，実際にプログラムの記述が容易化すること，下位ポリシの動的な導入が可能であることも確認する必要がある．

表 1 OSD コマンド
Table 1 OSD commands.

コマンド名	操作
create_group	グループの作成
remove_group	グループの削除
create	オブジェクトの作成
remove	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

2.4 ストレージシステムへの適用

提案方式を用いて，OSD ストレージシステムを実現した（以下，特にことわらない限り，本システム）．提案方式の適用方法について述べる．

OSD 規約は，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ストレージアクセス規約であり，次世代のストレージインタフェースとして注目されつつある．OSD ストレージシステムに対するアクセスコマンドを**表 1** に示す．OSD 規約は，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を定義している．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 サイズを指定することによって行われる．

2.4.1 スケジューリング方針

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備えている．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方針について考えてみる．

ディスク装置を効率良く運用するためには，単位時間あたりの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，1 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，アクセス頻度が高く，サイズの小さ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下位ポリシを実現し，上位ポリシが両者に対する主記憶割当てを，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．

2.4.2 各ポリシのアルゴリズム

本システムは，多数のクライアントから，動画の視聴等が行われることを想定しており，この場合，システムには高いスループット性能が要求される．ここでは，このような利用形態において良好なパフォーマンスを達成することを重視し，システムの評価値としてスループット（MB/s）を採用した．

提案方式は，複数のポリシを共存させながら，全体として 1 つのスケジューリング目標を達成するために動作する．ここでは，スループット性能を向上させることが全体目標になり，各下位ポリシは，これを達成するための 1 つのアルゴリズムとして動作する．よって，下位ポリシは，以上の全体目標を持つ上位ポリシに管理されることを前提に評価単位等を設計する必要がある．逆に，このようにして設計された下位ポリシは，異なるシステム全体の評価関数を持つ上位ポリシと組み合わせて利用することはできない．また，提案方式では，上下ポリシ間で扱われる資源は，各下位ポリシ間で排他的に利用される．

下位ポリシは，以下の 2 つを実現する．

(1) キャッシュポリシ
キャッシュページの置き換えを最適化することによって，キャッシュヒット率を高めるように動作する．単位時間あたりの，ページ単位のキャッシュヒット回数と総アクセス回数を荷重加算（本システムでは 19 : 1）した値を評価値とする．

(2) プリフェッチポリシ
データの先読み方針を決定する．ディスク上のファイル配置を考慮に入れてディスク I/O コストを最小化するように動作する．単位時間あたりのアクセスに関して，特定のサイズ（本システムでは 128 KB）で先読みした場合に必要となるシーク回数と比較して，実際に削減できたシーク回数を評価値とする．

以上のアルゴリズムを用いて，システムの運用を開始すると，システムへの負荷パターンに対応して，システムのスループットと下位ポリシの評価値の，様々な組合せが得られる．スケジューリング戦略表はこれらの値の組合せから構築される．上位ポリシはこの表を用いることで，様々な負荷パターンに対して，システム性能を向上させるための資源割当て方針を決定することができる．また，一定の周期で上位ポリシの資源割当てスケジューリングを行うことで，時間変化する負荷パターンにも対応することができる．

ここでは，以上のように，キャッシュページ置き換えと，データ先読み機能と，それぞれ異なる機能を持つ下位ポリシを実現したが，同じ機能についての下位ポリシを共存させることもできる．たとえば，キャッシュページ置き換えを，LRU アルゴリズムによって行うものと，LFU アルゴリズムによって行うもの等が考えられる．

3. 実　　装

本システムの構造を**図 3** に示す．OSD ストレージシステムと，これを利用する OSD クライアントは通常のネットワークで接続されており，TCP/IP 上に実装された専用の RPC プロトコルを用いて，OSD コマンドの伝達を行う．

ここでは，ポリシ階層化方式を汎用的に実現するためのフレームワーク（以下，ポリシ階層化フレームワーク）を，C++ 言語を用いて実装した．これにより，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

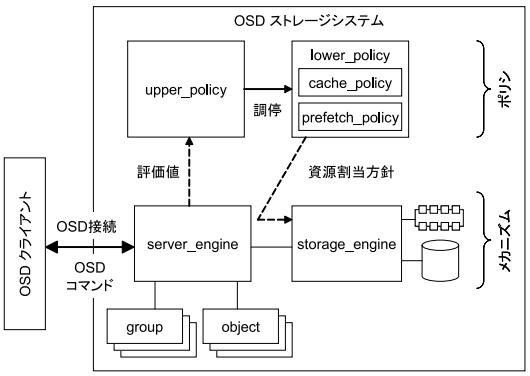

図 3 OSD ストレージシステムの構造
Fig. 3 Structure of OSD storage system.

表 2 フレームワークを構成するクラス
Table 2 Classes which constitute the framework.

root_resource	資源管理機構とのインタフェース．資源ごとに 1 つずつ作成される．有効資源量の観測，利用可能な資源量の上限設定等を行う
resource	資源管理機構とのインタフェース．ポリシごとに別個に作成される
lower_policy	下位ポリシのインタフェース
upper_policy	上位ポリシの実装

るので，一々考慮する必要がない．

本システムのソフトウェアは，

(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

から構成されている．

3.1 ポリシ階層化フレームワークの実装

表 2 に，ポリシ階層化フレームワークを構成する主要なクラスを示す．

lower_policy クラスは，下位ポリシを実装するためのベースクラスであり，上位ポリシの要求に応じて評価値，期待値，要求する資源の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て量の変更を通知するための関数等のインタフェースが定義されている．

upper_policy クラスは，上位ポリシの実装である．フレームワークが提供する他のクラスと異なり，実装自体が再利用可能になっている．個々のシステム実装においては，上位ポリシの評価値を得るための関数を 1 つ実装する必要がある．本システムでは，上位ポリシは 1 秒のスケジューリング周期で動作する．

upper_policy クラスの実装において，上下ポリシ間の評価値の相関関係の解析には三層バックプロパゲーションモデルのニューラルネットワーク[5] を用いた．これは各スケジューリング周期において，各下位ポリシの評価値を入力，上位ポリシの評価値を教師信号として学習を行い，また，各下位ポリシの期待値から，次回のシステム全体の評価値を推定するために利用される．2.2 節で述べたスケジュール戦略表は，実際には，ニューラルネットワークの学習データベースの形式で保持されている．

3.2 全体の実装

OSD ストレージシステムは，ポリシ階層化フレームワークを用いたポリシ群とメカニズムから構成される．

上位ポリシが評価値を得るための関数を実装した．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，storage_policy，cache_policy，prefetch_policy の 3 つを定義，実装した．storage_policy は，lower_policy を継承し，本システムにおける下位ポリシの共通インタフェースが定義されている．たとえば，OSD クライアントからの I/O 要求発生時に，OSD ストレージシステムのメカニズムが，下位ポリシへ資源割当て方針を問い合わせるためのインタフェースが含まれる．さらにこれを継承し，cache_policy，および，prefetch_policy を実装した．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．個々のファイルオブジェクトは，ある一時において，必ずいずれか 1 つの下位ポリシと対応付けられており，複数のポリシが同じブロックをキャッシュすることはない．storage_policy は，メカニズムがファイルオブジェクトとの対応付けの際に用いる，ポリシとファイルオブジェクトの適合度を返すインタフェースも含んでおり，本実装においては，それぞれのサブクラスは，単純にファイルサイズによって適合度を返す．対応付けは，この値を用いて，ファイルオブジェクトの新規作成時，新しい下位ポリシの追加時，各ポリシが既存ファイルオブジェクトとの対応付け解除を申請したときに，メカニズムによって行われる．

メカニズム部分は，ポリシ階層化フレームワークに含まれない．メカニズム部分は主に，group，object，server_engine，storage_engine のクラスから構成される．group，および，object は，それぞれ OSD 規約で定義されるグループとオブジェクトを表す．server_engine は OSD クライアントとの通信，group と object インスタンスの管理，OSD コマンドに対する資源割当て方針の下位ポリシへの問合せ，資源割当て方針の storage_engine への通知等の役割を持つ．また，上位ポリシが期待値を得るための関数を提供する．storage_engine は，ファイルシステムの記憶管理，バッファキャッシュ機構の役割を持ち，下位ポリシから指示された資源割当て方針を遂行する．二次記憶媒体であるハードディスクは，OS のバッファキャッシュ機構を無効化して，アクセスする．

OSD ストレージシステムにおいて構成されたスケジューラ全体の処理方式は，以下のようになる．OSD ストレージシステムに対してファイルアクセスを行うと，そのI/O 要求は OSD ストレージシステムのメカニズムが受信する．メカニズムは，I/O 該当部分にメモリページが割り当てられているかを確認し，キャッシュヒットした場合は，そのまま該当ページに I/O を行い，結果を OSD クライアントへ返信する．また，キャッ

シュミスした場合は，I/O 該当部分にページ割当てを行う必要性が発生するため，下位ポリシに，ページ置き換えスケジューリング要求を行う．このときにメカニズムから下位ポリシへ伝えられる情報には，(1) ファイルオブジェクト ID，(2) I/O の種類（Read または Write），(3) キャッシュミスを起こしたページオフセットが含まれる．ページ置き換えスケジューリング要求を受け取った下位ポリシは，各々のスケジューリングアルゴリズムによって，ページ置き換え命令リストを作成する．これは，その下位ポリシに対応付けられたファイルオブジェクト群に割り当てられているページの置き換えに関する命令のリストであり，それぞれの要素は，(1) ページ置き換え操作を行うファイルオブジェクト ID，(2) ページ置き換え操作の種類（ページ割当て，先読みを行うページ割当て，またはページ解放），(3) 操作を行うページオフセットの情報から構成される．最終的に，下位ポリシは，ページ置き換えスケジューリング要求に対する戻り値として，作成したページ置き換え命令リストをメカニズムへ渡し，メカニズムは，命令に従ってページ置き換えを行ったうえで，I/O を完了させる．個々のファイル I/O 要求にともなうページ置き換えは，個々の下位ポリシに割り当てられているページの総量の範囲内で行われる．下位ポリシに対して割り当てられているページの総量は，スケジューリング周期ごとに上位ポリシによって調節される．

4. 評　　価

提案方式の有効性について検討してみる．提案方式は，ポリシに注目してオブジェクトフレームワークを構築することにより，スケジューラ実装の構造化を図るとともに，実行時に漸近的に実装を改善することを可能にする方式である．以下，提案方式によって実現されたシステムに関して，方式が有効に機能していること，および，良好な性能が得られていることに焦点を当てて評価を行う．

システムは，Linux 2.6.12 を用いて，PC 上に構築した．PC のスペックは，CPU が AMD Athlon 64 3000+，メモリサイズが 1 GB，ハードディスクは WD2000JB（UATA/100，7200 rpm，8 MB バッファ）を用いた．

4.1 提案方式の有用性の評価

4.1.1 プログラム記述

提案方式を用いることにより，資源スケジューラの下位ポリシ部分を互いに独立性の高い複数のモジュールに分割できるようになる．**図 4** に，作成したプログラムの行数を示す．本システム中の 38%は，システムの種類に依存しないポリシ階層化フレームワークで構成できた．この部分は他の資源のスケジューリング機構にもそのまま適用できる．また，本システム中の 7%の部分は，本システム向けの下位ポリシで構成されており，これは互いに情報を共有しない 2 つの下位ポリシにほぼ等分されることが判明した．これまでの検討の範囲では，提案方式はプログラミングの工数削減，および，メンテナンスに有効であると結論できる．

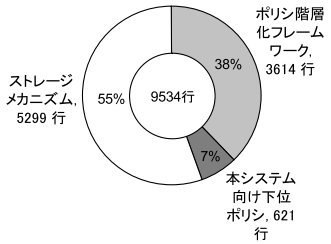

図 4 プログラム行数
Fig. 4 Number of program lines.

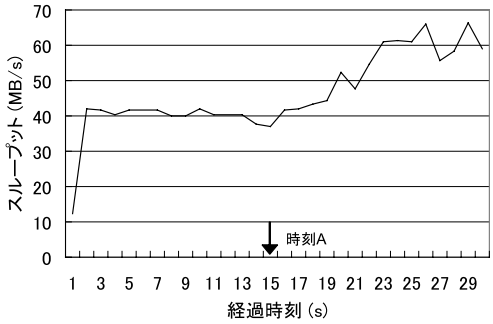

図 5 新しい下位ポリシの動的追加
Fig. 5 Dynamic addition of new lower policy.

4.1.2 新しい下位ポリシの動的導入

動作中のシステムに，新たな下位ポリシを追加できることを示す．ここでは，運用中の本システムに，いくつかのファイルを恒常的にメモリ上に保持する，RAM ディスクポリシを新たに追加した．**図 5** にその結果を示す．横軸が時間であり，縦軸がシステム全体のスループットを示す．

ここでは，新たに導入した下位ポリシが正しく動作していることを確認するために，RAM ディスクポリシに特に適合した負荷を用いて性能評価を行った．100 個のファイルをランダムにアクセスする．その際，50%のファイルのみ再参照を行うように設定されている．これらのファイルのみを優先的にメモリ上に置くことによってメモリの利用効率が改善される．時刻 0 から 2 つの既存下位ポリシを用いてランダムなファイルアクセスを開始し，時刻 A（15 秒）において，新

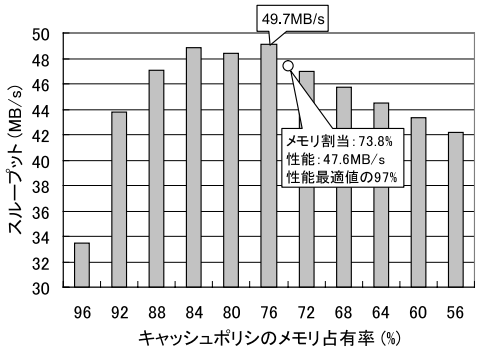

図 6 静的負荷パターンにおける性能特性
Fig. 6 Performance characteristic in a static load pattern.

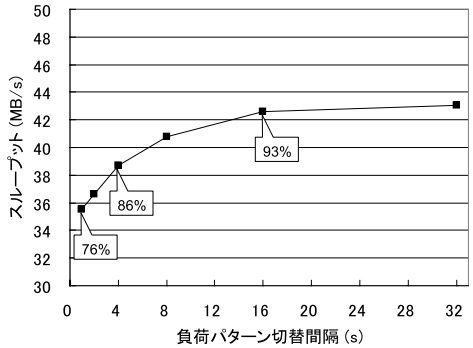

図 7 負荷パターン切替え間隔の違いによる性能特性
Fig. 7 Performance characteristic by the difference in a load pattern change interval.

たに RAM ディスクポリシを導入している．この下位ポリシの導入によって，システム性能が改善されることが観測された．これによって，実行時に動的にスケジューリングアルゴリズムを変更することが可能になったことが確認できる．

この機能を用いることにより，たとえば，特定の利用パターンに特化した性能最適化の実施が，スケジューラ全体を単一のプログラムによって実現する方式より容易に行えるようになる．

4.2 性能の評価

4.2.1 定常負荷における性能

定常的な負荷に対して，キャッシュ機構とプリフェッチ機構に対する資源割当てが適切に行われ，条件ごとに最適な性能が得られていることを確認する．ここで，評価に用いる負荷として，大きさの異なる複数のファイルを長時間ランダムにアクセスするものを用いる．アクセス対象となるファイルの大きさの分布が，負荷の性質を決める．以下では，これを負荷パターンとよぶことにする．定常的な負荷パターンでは，ファイルの大きさの分布はつねに一定である．アクセスは 8 KB ずつ Read 要求を行い，ファイルの最初から最後まで読み終わったら，次のファイルをランダムで選択する．これを 10 スレッド，並列して動作させ，負荷パターンを発生させた．

図 6 は，キャッシュポリシとプリフェッチポリシに対するメモリ割当て量を固定にして，1 MB から 100 MB までの 90 個ファイルをランダムにアクセスした際のスループットを示したものである．複数のメモリ割当て量のケースについて示してある．縦軸がシステムのスループットであり，横軸が，キャッシュポリシに対するメモリ割当て量を全メモリに対する比の形式で示している．全メモリの 76%をキャッシュポリシに割り当てた際に最大性能 49.1 MB/s のスループットが得られている．これが，この定常負荷における最適性能値となる．

これに対して，同一の負荷パターンにおいて，上位ポリシを動作させて動的にメモリ割当て量を決定したときに得られたメモリ割当て量と性能値を，白抜きの円で示している．メモリ占有率の平均は 73.8%となり，性能の時間平均値では 47.6 MB/s のスループットが得られた．最適値に対する性能の低下は，おもに，ランダムなアクセスによる割当て量の揺らぎによって生じたものである．他の測定条件においても，最悪のケースで性能最適値の 91%以上の性能が得られている．

4.2.2 時間変化する負荷における性能

負荷の性質が時間によって変化した場合でも，下位ポリシに対する資源割当てを適切に変更して最適な性能が得られるように上位ポリシが動作していることを確認する．

図 7 は，実現したシステムに対して，2 種の負荷パターン A，B を繰り返し与えたときの性能特性を表している．縦軸がシステムのスループットであり，横軸が負荷パターン A，B を繰り返す時間間隔を表している．たとえば，間隔が 1 秒の場合，負荷パターン A を 0.5 秒，B を 0.5 秒実行することを繰り返す．負荷パターン A と B は，それぞれ特定の負荷パターンを持つ，10 のスレッドを並列して動作させて発生させた．それぞれのスレッドのアクセスは 4.2.1 項と同様に行う．2 種の負荷パターンを別々に測定した際のスループットは，負荷パターン A が 36.5 MB/s，B が 55.1 MB/s であった．ここから，負荷パターンの動的変動による性能ロスを無視した理想的な性能は，45.8 MB/s である．パターン切替え間隔が 16 秒では，理想性能の 93%の性能が得られた．これに対して，4 秒では 84%，1 秒では 78%となった．

ここから，ある負荷パターンの持続時間が，スケ

ジューリング周期に対して十分大きい場合には，良好な性能が得られることが分かった．たとえば，デスクトップシステムにおける個々のコマンド実行や，オンライントランザクション処理システムにおける個々のトランザクション実行等では，個々の処理のターンアラウンド時間が短いため，これらの負荷による変化には追従しないが，マルチメディアファイルの視聴や，バッチジョブの実行等のように，一定の負荷パターンが継続して発生する処理に対して性能最適化を図る目的には有効に機能することが推定される．

4.2.3 総合的な性能

実現したシステムの総合的な性能について評価する．本システムの典型的な利用形態として，通常ファイルやマルチメディアファイルをアクセスする複数の端末に対するファイルサーバとして運用することがあげられる．既存システムの運用状況を観測し，OSD コマンド列の形式で負荷を生成するベンチマークプログラムを開発して性能を比較した．

このベンチマークプログラムは，Apache HTTP Server のソースプログラムのビルド，および，動画コンテンツの視聴を同時に行うことによって生ずるストレージアクセスの負荷パターンを，OSD コマンド列の形式で生成する機能を持つ．ビルド作業は，211 回のコンパイル処理を含んでいる．動画コンテンツは，デジタルハイビジョンを想定し，25 Mbps のスループットによる OSD I/O パターンを用いている．コンパイル負荷においては，実負荷で生じた各コマンド間の待ち時間を再現する．ストリーム負荷においては，ブロック単位で読み込みを行いながら，指定したスループットを超えそうな場合は，適時，ブロック間に待ち時間を挿入して調整を行う．ベンチマークプログラムの動作時には，これらの負荷をそれぞれ，任意の数の異なるスレッドを用い，並列して発生させることができるようになっている．各スレッドは，それぞれ異なるファイルセットにアクセスを行い，1 つのコマンドが完了したら，必要に応じて待ち時間をはさんでから次のコマンドを発行する形で動作する．

性能比較の対象として，OSD ストレージシステムを，EXT2 ファイルシステムを用いて別途実現した．OSD のオブジェクトを EXT2 上のファイルに対応付けて，OSD コマンドを実行する．EXT2 上のファイルはあらかじめオープンされており，オープンのためのオーバヘッドは生じない．以下，提案方式によるシステムを通常版，EXT2 を用いたシステムを EXT2 版とよぶ．

性能評価中に EXT2 版がバッファキャッシュに消費したメモリ量を観測し，通常版もそれと同じメモリ量で運用することによって最大メモリ消費量が同じになるようにしている．

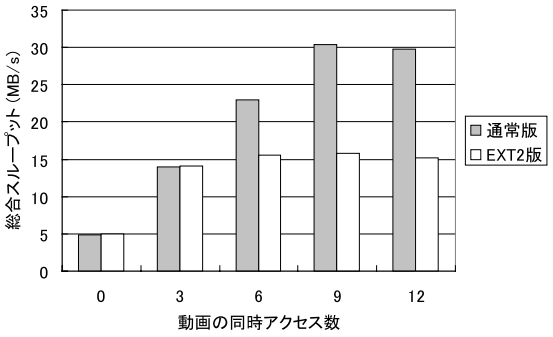

図 8 動画の同時アクセス数の違いによる性能特性
Fig. 8 Performance characteristic by the difference in the number of streaming threads.

測定結果を**図 8** に示す．横軸は，動画視聴に起因するストリーミングファイルアクセス（以下，動画アクセス）を何個並列に行っているかを示しており，それぞれについて，通常版と EXT2 版の性能を測定している．縦軸は，複数のビルド作業，および，動画アクセスに対する総合スループットの平均値を示している．ビルド作業は，いずれの測定においても，10 個並列して実行している．

動画の同時アクセス数が 0 から 3 個の場合は，EXT2 版がわずかに高い性能を得たが，ほとんど性能差は見られなかった．それぞれ，通常版は EXT2 版の，96.9%，98.7%の性能を得ている．

動画の同時アクセス数が 6 個以上になると，EXT2 版の性能は 15.5 MB/s で飽和した．これに対して，通常版の性能は，動画同時アクセス数が 9 以上になってから 30 MB/s で飽和した．動画同時アクセス数が 9 以上の条件では，通常版は EXT2 版の 194%程度の性能を得た．

動画同時アクセス数が 6 以上の場合における EXT2 版の性能飽和は，動画アクセスによってキャッシュ内容が全面的に書き換えられてしまうことに起因するものと考えられる．並行して実行されているビルド処理では，たとえば，中間ファイルに関して，キャッシュ中のファイルを再参照する可能性が高い．キャッシュ書き換えによって廃棄されたページが再参照されると，ディスク I/O にともなうシーク動作が必要になり，再参照部分のサイズ自体は小さくても，システムの総合スループットに大きな悪影響を及ぼす．実際に，EXT2 版において，9 個の動画同時アクセスのみの負荷と，ビルド処理による負荷の，2 つのベンチマーク

を別個に実施したスループット値を足し合わせた値は33.1 MB/s となり，通常版で両者を同時に実行した際のスループットにほぼ等しくなっている．これから，EXT2 版の性能飽和が，2 種類の負荷の競合によるものと推測される．

提案方式によるシステムでは，キャッシュに用いる主記憶領域と，ストリーミングデータアクセスに用いる主記憶領域が自動的に適切なサイズ比率で分離されることによって，性能低下が防がれた．この結果から，マルチメディアファイルのアクセスと，通常作業が混在する環境では，提案方式が有効に作用することが推定できる．

4.3 CPU オーバヘッドの評価

4.3.1 通常版と EXT2 版における CPU オーバヘッドの比較

提案方式を用いたシステムでは，上位ポリシにおいては，資源配分スケジューリングや学習を行い，各下位ポリシにおいては，評価値や期待値等の自己評価機構が必要になることから，既存方式と比較してオーバヘッドが生じることが予想される．ここでは，4.2 節で行った測定における，通常版と EXT2 版の CPU 時間を測定し，ここから 1 GB のデータ転送を行うごとに必要となる CPU 時間を求めた．測定した負荷パターンは，動画の同時アクセス数が 3（パターン A），6（パターン B），9（パターン C）の場合の 3 パターンを採用した．これには，システム初期化時と RPC に必要な CPU 時間は含まない．測定結果を**表 3** に示す．

測定を行った結果，通常版の方が，CPU を多く消費していることが分かった．特にパターン C においては，1 GB のデータ転送を行うために，通常版は EXT2 版の 1.7 倍にあたる 6.27 秒もの CPU 時間を使用している．ここから，提案方式を適用する場合，割当て対象となる資源が CPU である場合や，システムを運用する計算機の CPU 使用率に余裕がない場合は，CPU がボトルネックとなり期待した性能が得られない可能性がある．

4.3.2 下位ポリシ追加時の CPU オーバヘッド

提案方式では，任意個の下位ポリシに対して，一定の周期で資源割当てスケジューリングを行う．この際，下位ポリシの数に応じて，資源割当て案の検討やスケジューリング戦略表の構築が行われる．このため，下位ポリシ追加にともない，オーバヘッドが生じると考えられる．ここでは，下位ポリシ数別に，上位ポリシのスケジューリングを 1000 回行ったときの CPU 時間を測定し，スケジューリング 1 回あたりの平均 CPU 時間（ミリ秒）を求めた．スケジューリング戦略表は十分に学習が行われており，学習によるコストは生じない．また，各下位ポリシの評価値，期待値の決定処理にともなう CPU 時間は含まれない．測定結果を**表 4** に示す．測定結果より，スケジューリング 1 回あたりの CPU 時間は，ポリシが 1 つ増えるごとに，4.7 倍程度に増加していくことが分かった．

表 3 転送量あたりの CPU 時間
Table 3 CPU time per storage traffic.

パターン	通常版（sec/GB）	EXT2 版（sec/GB）
パターン A	4.23	2.73
パターン B	4.99	4.15
パターン C	6.27	3.77

表 4 転送量あたりの CPU 時間
Table 4 CPU time for upper policy scheduling.

下位ポリシの数	CPU 時間（msec）
2	0.655
3	2.985
4	14.223
5	68.070
6	315.734

ポリシ数を n とすると，上位ポリシは，スケジューリングごとに 4^n 個の資源割当て案を作成し，採用するかどうかの評価を行っている．この処理が CPU 時間の増加に関する主な要因であると考えられる．残りの 0.7 倍の値は，ポリシ数増加によって，スケジューリング戦略表が複雑になることや，スケジューリング制御が複雑になること等が理由として考えられる．これより，たとえばスケジューリング周期が 1 秒の場合は，下位ポリシの数は 5 個程度に抑えることが望ましいことが分かった．

5. 従来の研究

5.1 オペレーティングシステムの資源管理機構の拡張方式

オペレーティングシステムの資源管理機構を拡張可能にする試みは，従来から検討が続けられてきた，本稿での試みと最も関連の深い分野である．重要なものとして，以下があげられる．

(1) 実行時に管理機構をダイナミックにロードできるようにする試み

K42[6]，Vino[7]，Spin[8] 等のオペレーティングシステムでは，資源管理機構が実行時に核内にロードできるようになっている．また，Vassal[9] では，WindowsNT オペレーティングシステムにこのような機能を追加することを試みている．また，AIX[10] や Linux[11] のダイナミックリンク機構もこのよう

な機能の実現に利用することができる．

(2) ユーザレベルで資源管理機構を拡張できるようにするシステムインタフェースの検討
最近では，マイクロカーネル Lavender に 2 レベルスケジューラを構成する試み[12] や，Infokernel[13] や Exokernel[14] における試みがあげられる．文献 12) では，ローダブルカーネルモジュールとして実装された複数のスケジューリングポリシを共存して運用するとともに，調整ポリシを設けて各ポリシによるスケジューリングの競合を解決している．このとき，提案方式と異なり，調整ポリシはシステムごとに実装される．Infokernel では，核内のポリシの情報をユーザプログラムに公開する機構を考案し，アプリケーションプログラムがスケジューリングポリシを変更できるようにしている．

(3) スケジューラを実現する専用言語の開発
たとえば，Bossa[15] では，プロセスのスケジューラを設計するための専用言語である DSL を開発して用いている．

これらの試みの主眼は，種々のアプリケーションの性質に適合するように資源管理機構を変更することにある．その際に，効率や安全性に問題が生じないように種々の機構が考案されている．たとえば，文献 12) や，Infokernel に見られるように，スケジューリング機構全体ではなく，ポリシのみを変更できるようにすることが試みられてきた．

特定目的に特化したポリシを利用する際の問題は，システム全体としての最適化が行いにくくなり，資源の専有化等の問題が起こりやすくなる点があげられる．これを防ぐために，たとえば，Vino では，拡張部分をトランザクションとして実行している．しかしながら，システム全体としての最適化を保障するものにはなっていない．

本稿で新たに試みたのは，実行時に導入されたポリシの評価機構（上位ポリシ）を設け，その評価結果に応じて資源を割り当てるようにした点にある．これによって，システム全体の性能の観点から，個々のポリシを適切に運用できるようにした．また，この評価を行いやすくするための機構として，ポリシの評価値に関する期待値を自己申告するようにした．

評価機構の実現において，学習機構を用いた．資源管理に学習機構を利用することは広く試みられている．たとえば文献 16) では，分散ストレージシステムにおける記憶階層間でのデータ移動に際して，強化学習を用いてポリシの最適化を図る方法について議論している．提案方式では，上位，下位ポリシ間の連携によって学習が単純化され，強化学習ではなく，教師信号付き学習によって評価機構を構成することが可能になった．

5.2 複数のアルゴリズムの組合せによる動的最適化方式

複数のアルゴリズムを用いて動的な性能最適化を行う試みは，超流動 OS[17] のアルゴリズム・アダプテーションで行われている．この方式は，ある問題を解くのに複数のアルゴリズムが使用できるとき，引数となるデータの特徴から，最適なアルゴリズムを自動的に選択する試みである．提案方式と比較すると，この方式は，最も性能最適化が期待できるアルゴリズムを，最終的に 1 つ選択する方式であるのに対して，提案方式は，複数のスケジューリングアルゴリズムを同時に共存させながら運用し，資源割当てバランスを調整することで性能最適化を行う方式であるという違いがある．

5.3 ストレージシステムの性能最適化

本稿では，ストレージシステムの性能改善を試みた．ストレージシステムの性能に関する技術動向全体を俯瞰することは困難であるので，新しい方向性についてのみ簡単に触れる．ストレージ上のプロセッサを，アプリケーション実行やアプリケーションの性質を考慮した性能最適化に利用することが試みられた．Active Disks[18]，Active Storage[19] における試みが知られている．また，ストレージをネットワークに接続して利用することが広く行われるようになっている．OSD 規約は双方の試みに対応して，広く用いることができる標準的なインタフェースを定義したものである．提案方式は，OSD ストレージシステムの性能最適化に適合すると考えられる．たとえば，OSD と同様の枠組みを利用して，オートノミックストレージ技術の検討が行われている．これは，たとえばディスクに対するデータアクセスのパターンに関してデータマイニング技術を利用して性能の最適化を図る試み[20] である．このデータマイニングを，提案方式におけるポリシを用いて実現することによって，システムの変更部分が限定され，システム開発効率が改善されることが期待できる．

6. 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．汎用的にポリシ階層化方式を実現するフレームワークを開発し，これを利用して OSD ストレージシステムを構築した．

本システムに対して，ポリシ階層化方式の有効性，性能の観点から評価を行い，ポリシ階層化方式を用いることで，資源スケジューラの柔軟な構築が容易になり，また，適切な資源割当てによって良好な性能を得られることが確認できた．

著者らが開発した大規模分散ファイルシステムのファイルキャッシュ機構の性能改善に本システムを適用する予定である．さらに，複数キャッシュ間でのファイル移動や，分散ファイルシステムと連動したジョブ実行スケジューリングにも提案方式を適用し，より広範なスケジューリング機構への適用可能性を検討してゆきたい．

参 考 文 献

1) Weber, R.O.: Technical Information technology—SCSI object-based storage device commands (OSD), *Council Proposal Document T10/1355-D, Technical Committee T10* (2002).
2) 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，情報処理学会研究報告，OS-100, Vol.2005, No.79, pp.25–32 (2005).
3) 小柳順裕，田胡和哉，山下直人：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，情報処理学会研究報告，OS-103, Vol.2006, No.86, pp.63–70 (2006).
4) Levin, R., Cohen, E., Corwin, W., Pollack, F. and Wuld, W.: POLICY/MECHANISM SEPARATION IN HYDRA, *Proc.5th ACM Symposium on Operating Systems Principles* (1975).
5) 豊田秀樹：非線形多変量解析—ニューラルネットによるアプローチ，朝倉書店 (1996).
6) Silva, D.D., Krieger, O., Wisniewski, R.W., Waterland, A., Tam, D. and Baumann, A.: K42: An Infrastructure for Operating System Research, *ACM SIGOPS Operating Systems Review*, Vol.40, No.2 (2006).
7) Seltzer, M.I., Endo, Y., Small, C. and Smith, K.A.: Dealing with disaster: Surviving misbehaved kernel extensions, *Proc. 2nd Symposium on Operating Systems Design and Implementation*, pp.213–227 (1996).
8) Bershad, B., Savage, S., Pardyak, P., Sirer, E.G., Becker, D., Fiuczynski, M., Chambers, C. and Eggers, S.: Extensibility, Safety and Performance in the SPIN Operating System, *Proc. 15th ACM Symposium on Operating Systems Principles*, pp.267–284 (1995).
9) Candea, G.M. and Jones, M.B.: Vassal: Loadable Scheduler Support for Multi-Policy Scheduling, *Proc. 2nd USENIX Windows NT Symposium*, pp.157–166 (1998).
10) IBM: IBM AIX 5L UNIX operating system. http://www-03.ibm.com/servers/aix/
11) Linux Kernel Organization: The Linux Kernel Archives. http://www.kernel.org/
12) 毛利公一，大久保英嗣：マイクロカーネルLavenderにおける2レベルスケジューラの構成方式，情報処理学会論文誌，Vol.40, No.6, pp.2534–2542 (1999).
13) ArpaciDusseau, A.C., ArpaciDusseau, R.H., Burnett, N.C., Denehy, T.E., Engle, T.J., Gunawi, H.S., Nugent, J.A. and Popovici, F.I.: Transforming Policies into Mechanisms with Infokernel, *Proc. 19th ACM Symposium on Operating Systems Principles*, pp.90–105 (2003).
14) Engler, D.R., Kaashoek, M.F. and O'Toole, J.: Exokernel: An operating system architecture for application-level resource management, *Proc. 15th ACM Symposium on Operating Systems Principles*, pp.251–266 (1995).
15) Barreto, L. and Muller, G.: Bossa: A Language-based Approach for the Design of Real Time Schedulers, *Proc. 10th International Conference on Real-Time Systems* (*RTS'2002*), pp.19–31 (2002).
16) Vengerov, D.: Dynamic Tuning of Online Data Migration Policies in Hierarchical Storage Systems using Reinforcement Learning, Sun Microsystems Technical Report SMLI TR-2006-157 (2006).
17) 平野 聡，田沼 均，須崎有康，浜崎陽一，塚本享治：超並列システム用オペレーティングシステム「超流動OS」の構想，情報処理学会研究報告，1992-OS-058, Vol.1993, No.27, pp.17–24 (1993).
18) Uysal, M., Acharya, A. and Saltz, J.: Evaluation of Active Disks for Decision Support Databases, *Proc. 6th International Symposium on High Performance Computer Architecture* (*HPCA'00*), pp.337–348 (2000).
19) Riedel, E., Gibson, G. and Faloutsos, C.: Active Storage for Large-Scale Data Mining and Multimedia Applications, *Proc. 24th Int. Conf. Very Large Data Bases* (*VLDB*), pp.62–73 (1998).
20) Li, Z., Chen, Z., Srinivasan, S.M. and Zhou, Y.: C-Miner: Mining Block Correlations in Storage Systems, *Proc. USENIX Conference on File And Storage Technologies* (*FAST*), pp.173–186 (2004).

(平成18年8月28日受付)
(平成19年2月 2 日採録)

小柳 順裕（学生会員）

2004 年東京工科大学工学部情報工学科卒業．同年（株）エヌ・ティ・ティ・データ・フロンティア入社．2007 年東京工科大学大学院バイオ・情報メディア研究科修士課程修了．同年 4 月より，慶應義塾大学大学院理工学研究科博士後期課程在学中．オペレーティングシステムの研究に従事．

田胡 和哉（正会員）

1986 年筑波大学大学院工学研究科博士課程修了．工学博士．筑波大学電子情報工学系助手，東京大学工学部助手，日本 IBM 東京基礎研究所を経て，現在東京工科大学コンピュータサイエンス学部教授．オペレーティングシステムの構成方式に興味を持つ．1984 年情報処理学会論文賞．ACM 会員．

市村 哲（正会員）

1989 年慶應義塾大学理工学部計測工学科卒業．1994 年同大学大学院理工学研究科博士後期課程修了．博士（工学）．同年富士ゼロックス（株）入社．1997〜1999 年富士ゼロックスパロアルト研究所（FXPAL）駐在．2002 年より東京工科大学助教授．グループウェア，ネットワークサービス，生体情報活用等の研究に従事．『IT TEXT 基礎 Web 技術』，『IT TEXT 応用 Web 技術』（オーム社）．DICOMO 2003，2005，2006 優秀論文賞受賞．ACM，日本 VR 学会，電子情報通信学会各会員．

松下 温（フェロー）

1963 年慶應義塾大学工学部電気工学科卒業．1968 年イリノイ大学大学院コンピュータサイエンス専攻修了．工学博士．1989 年より 2002 年 3 月まで慶應義塾大学理工学部教授，2002 年 4 月より東京工科大学教授および慶應義塾大学理工学部客員教授，2003 年 4 月より 2006 年 3 月まで東京工科大学コンピュータサイエンス学部学部長および慶應義塾大学理工学部客員教授，2006 年 4 月より住宅情報化推進協議会．マルチメディア通信，コンピュータネットワーク，グループウェア等の研究に従事．情報処理学会理事，同学会副会長，マルチメディア通信と分散処理研究会委員長，グループウェア研究会委員長，電子情報通信学会情報ネットワーク研究会委員長，MIS 研究会委員長，バーチャルリアリティ学会サイバースペースと仮想都市研究会委員長，情報処理学会 ITS 研究会委員長等を歴任．郵政省，通産省，建設省，農水省，都市基盤整備公団，行政情報システム研究所等の委員長，座長，委員等を多数歴任．特に国土交通省，住宅情報化標準策定委員会委員長，経済産業省総合エネルギー調査会電子計算機と磁気ディスク委員会委員長を務める．現在，経済産業省総合エネルギー調査会ルータ装置基準委員会委員長，最高裁判所専用委員．『やさしい LAN の知識』（オーム社），『201x 年の世界』（共立出版）等著書多数．1993 年情報処理学会ベストオーサ賞，1995 年および 2000 年情報処理学会論文賞，2000 年 10 月 20 日情報処理学会 40 周年記念 90 年代学会誌論文賞，2000 年 10 月 2 日電子情報通信学会フェロー，2000 年 10 月 VR 学会サイバースペース研究賞，2001 年 5 月情報処理学会功績賞，2002 年 3 月情報処理学会フェロー，電子情報通信学会，人工知能学会，ファジィ学会，IEEE，ACM 各会員．