オペレーティング・システム　58－2
（１９９３．３．１９）

オブジェクト堆積モデルに基づくファイル・サーバの実現

苅部 朋幸、新城 靖、清木 康

〒３０５　茨城県　つくば市　筑波大学　電子・情報工学系
電話：０２９８－５３－５１６３　　Ｆａｘ：０２９８－５３－５２０６
電子メール：{karibe, yas}@softlab. is. tsukuba. ac. jp, kiyoki@is. tsukuba. ac. jp

この論文は、オブジェクトの堆積というモデルに基づき、高度な機能を提供するファイル・サーバの実現方式について述べている。オブジェクトの堆積とは、object-basedシステムにおいて、様々なオブジェクトの機能を統合して利用するためのモデル化の手法である。object-based分散型オペレーティング・システムでは、オブジェクト識別子と遠隔手続き呼出しにより、一様なインタフェースを持つオブジェクトが統合される。この論文は、分散型オペレーティング・システムにおける、フィルタ機能を持つ堆積可能オブジェクトを提供するサーバの実現方式を提案している。堆積可能オブジェクトとは、自分自身と同等なインタフェースと機能を持つオブジェクトの識別子を保持し、かつ、自分自身の機能を実現するために、必ずそのオブジェクト識別子を用いてそのオブジェクトを呼び出すようなオブジェクトである。フィルタの例としては、文字列の置換え、圧縮、選択、ソートがあげられる。特徴的なサーバの１つは、外部のプログラムを利用してフィルタ機能を実現している。もう１つは、ファイル単位のキャッシングを利用している。キャッシングの利用により、行単位やファイル単位のフィルタの実現が容易になり、さらに、性能も改善される。サーバ内部の並列処理と並行性制御の実現のために、軽量プロセスが利用されている。提案方式に基づき、３つのサーバが実現され、その性能が示され、そして、実現方式の有効性が論じられている。

Implementation of File Servers Based on the Object-Stacking Model

Tomoyuki Karibe, Yasushi Shinjo and Yasushi Kiyoki

Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba, Ibaraki 305, JAPAN
Phone: +81 298 53 5163 Fax: +81 298 53 5206
Email: {karibe,yas}@softlab.is.tsukuba.ac.jp, kiyoki@is.tsukuba.ac.jp

This paper presents implementation methods of file servers which provide high-level functions based on the object-stacking model. Object-stacking is a model for integrating the functions of several objects in object-based systems. In object-based distributed operating systems, objects which have uniform interfaces are connected by object identifiers and remote procedure calls. This paper shows implementation methods of servers which provide stackable objects with filer functions. A stackable object holds the identifier of another object, and calls the object with the identifier to implement its function. Examples of filters are string substitution, selection, compression and sorting. A characteristic server uses filter programs outside it, and provides stackable objects. Another uses caches of whole files. Use of caches makes it easy to implement filters for lines and for whole file. Lightweight processes are used to realize parallel processing and concurrency control. Several servers have been implemented based on the proposed methods. In this paper, performance of the servers is shown, and effectiveness of the proposed method is discussed.

1　はじめに

最近の分散型オペレーティング・システムの多くは、object-basedシステムとして構築されている［2］。そのようなシステムでは、カーネルは、プロセス間通信と物理入出力機能のみを提供し、カーネル外のサーバがシステムの機能の大部分を提供する。カーネルが提供するプロセス間通信プリミティブとしては、遠隔手続き呼出しが広く利用されている。

一方、ＵＮＩＸのパイプは、複数のプログラムが持つ機能を統合して利用するための仕組みとして、最も成功したものの１つである。ＵＮＩＸでは、パイプにより結合して利用可能なプログラムは、フィルタ（filter）と呼ばれる。各フィルタは、それぞれソート（sort）、文字列の置き換え（sed）、選択（grep）といった、単純な機能を提供する。これらのフィルタは、パイプにより結合され、複雑なフィルタを形成する。

オブジェクトの堆積（object-stacking）は、object-basedシステムにおいて、サーバが提供するオブジェクトの機能を統合するためのモデル化の手法である。オブジェクトの堆積では、個々のサーバは、単純な機能を持つオブジェクトを提供する。それらのオブジェクトは、オブジェクト識別子と遠隔手続き呼出しにより結合され、それらの機能を合せ持つようなオブジェクトを形成する。この論文では、フィルタ機能を持つオブジェクトを提供するサーバの実現方式について述べる。

本論文では、サーバの開発時の効率の改善と実行時の資源の有効利用を図る方法として、１つのサーバに複数の機能を持たせる方法を提案する。オブジェクトの堆積では、１つのオブジェクトは、１つの機能だけを提供する。単純に機能ごとにサーバを構築したならば、非常に多くのサーバが必要となる。この結果、サーバを開発する手間が増大し、実行時には、大量の資源が必要となる。この問題を解決するために、本論文では、複数の機能を提供するサーバを用いる方法を提案する。

本論文では、外部のプログラムを利用して、フィルタ機能を実現する方式を提案する。この方式の特徴は、サーバ内部にフィルタ機能を実現する方法と比較して、拡張性が高い点にある。本論文では、ファイル単位のキャッシングを利用する方式を提案する。この方法では、ソートなどのファイル全体を単位とするフィルタ処理や、選択などのファイルの大きさを変化させるフィルタ処理の実現が容易になり、さらに、実行効率が改善される。ここで述べるサーバは、内部に、クライアントからの要求を処理する部分や、フィルタ処理のためのデータの入力を行う部分といった並列処理の単位を多数含んでいる。これらの並列処理の単位を軽量プロセスとして実現している。そして、オブジェクトの相互排除を、軽量プロセス間の同期として実現している。

２章では、オブジェクトの堆積と名前サーバについて述べる。３章では、フィルタの性質について論じる。４章では、サーバの３通りの実現方式を示す。５章では、４章で述べる方式に基づいて実現したサーバの性能について述べる。６章では、まとめを行う。

2　オブジェクトの堆積

オブジェクトの堆積（object-stacking）の基本的な考え方は、一様なインタフェースを持つオブジェクトを積み重ねることである。オブジェクトとは、手続きとデータをカプセル化したものであり、公開された手続きによってのみ内部のデータの操作が許される。分散型オペレーティング・システムでは、オブジェクトは、サーバにより管理されている。インタフェースとは、オブジェクトが受け付けることができる手続きの集合である。オブジェクトに対する手続きをオブジェクト手続き、サーバに対する手続きをサーバ手続きとよぶ。**積み重ねる**（to stack）とは、上位層のオブジェクトが下位層のオブジェクトの識別子を保持し、かつ、上位層のオブジェクトが自分自身の機能の実現するために下位層のオブジェクトを呼び出すことである。

図１に、３つのファイル・オブジェクトが積み重ねられている様子を示す。上の２つのオブジェクトは、**堆積可能オブジェクト**（stackable objects）であり、それぞれ下位層のオブジェクトの識別子を保持している。最下位層のオブジェクトは、**基底オブジェクト**（a bottom object）であり、他のオブジェクトの識別子を保持していない。堆積可能オブジェクトのサーバを堆積可能サーバ、基底オブジェクトのサーバを基底サーバと呼ぶことにする。積み重ねられたオブジェクト全体を１個のオブジェクトとみなすことができる場合、それを**堆積オブジェクト**（a stack object）と呼ぶ。堆積オブジェクトの最上位層のオブジェクトを、頂上オブジェクト（a top object）と呼ぶ。

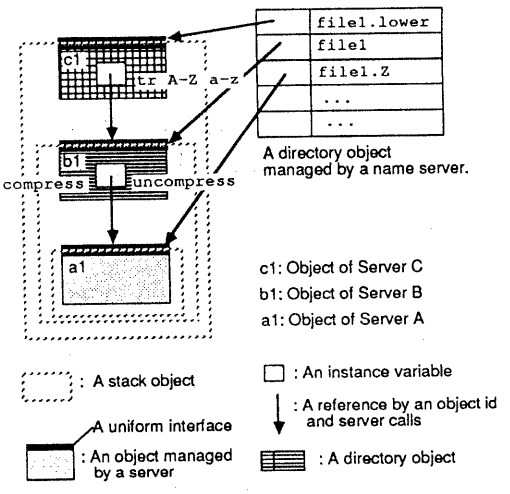

図１　堆積オブジェクトとディレクトリ・オブジェクト

図１の例では、上から次のような機能を持っている。

- 大文字を小文字に変換する
- データを圧縮／解凍する
- データを貯蔵する

そして、これらのオブジェクトの機能が組み合わせられ、読み込むと圧縮されたデータを解凍し、大文字を小文字に変換するファイル・オブジェクトが作られている。

図1において、クライアントから読込み要求を受け付けると、堆積可能サーバは、クライアントとして、対応する下位層のオブジェクトのサーバに読込み要求を送る。次に、読み込んだデータに対して、大文字を小文字に変換する、あるいは、圧縮されたデータを解凍する等の処理を行う。そして、処理を行った結果をクライアントに返す。基底サーバは、通常のファイル・サーバと同様に、ディスク・ブロック等からデータを読み込み、他のサーバを呼び出すことなく処理を行い、結果をクライアント返す。図1のようにオブジェクトが積み重ねられた場合、2つの堆積可能オブジェクトと1つの基底オブジェクトの機能が統合される。

2．1　名前サーバによるデータへの名前付け

オブジェクトの堆積において、名前サーバ（name server）は、オブジェクトの文字列の名前を識別子に変換するサーバである。そのために、オブジェクト識別子とオブジェクトの文字列の名前の組のリスト（ディレクトリ）を保持している。

オブジェクトの堆積は、利用者に対して処理（機能、手続き、関数）ではなく、データ（オブジェクト）に名前を付ける能力を提供する。ＵＮＩＸでは、フィルタは、（名前が付けられた）プログラムであり、データではない。利用者は、シェル・スクリプト（shell script）の作成や別名（aliases）の設定を行うことで、（名前がある）フィルタを組み合わせたものに名前を付けることができる。これに対して、オブジェクトの堆積では、処理が施された結果に対して名前を付けることが可能となる。そして、それらのオブジェクトを元に、新たにオブジェクトを生成することができる。

データに対する名前付けの利点として、サーバによる自動的な一貫性の保証が可能になることがあげられる。たとえば、あるファイルのデータがソートされたものが必要な場合を考える。ＵＮＩＸでは、利用者は、ソート・プログラムを明示的に起動し、ソートされたデータを作成することになる。そして、元のファイルのデータが更新される度に、利用者は、自らソート・プログラムを実行する必要がある。（あるいは、それを補助するプログラム make を明示的に実行する必要がある。）これに対して、オブジェクトの堆積では、元のファイルのデータをソートしたものをファイルとして名前を付けることができる。そして、利用者は、その名前が付けられたファイルを読み込むことで、一貫性が保証されたデータを自動的に得ることが可能となる。

2．2　他の研究との比較

モデルとしてのオブジェクトの堆積と他の研究との比較については、文献［4］において詳しく述べた。継承と比較して、オブジェクトの堆積の特徴は、堆積可能オブジェクトの存在である。継承では、下位層のオブジェクトという概念が存在しない。また、文献［1］では、ＮＦＳのＶｎｏｄｅを一様なインタフェースとして、層を構築する方法が提案されている。この方法と比較して、オブジェクトの堆積の特徴は、オブジェクトごとに堆積が作られること、および、オブジェクト識別子と遠隔手続き呼出しを利用して、異なるアドレス空間にあるオブジェクトが結合されることにある。

3　フィルタ

本章では、4章で述べるサーバで扱うフィルタの分類を行う。

（1）逆関数の存在の有無

フィルタは、逆関数の存在の有無により2種類に分類される。1つは、逆関数が存在するフィルタであり、例として、データの圧縮と解凍、暗号化と復号化が挙げられる。もう1つは、逆関数の存在しないフィルタであり、ソートや選択などがある。

（2）フィルタ処理の単位

フィルタは、処理の単位により、2種類に分類される。1つは、フィルタ処理がバイト単位で行われるものである。このようなフィルタとしては、英大文字を英小文字に変換するものが挙げられる。もう1つは、フィルタ処理が行単位やファイル単位で行われるものである。このようなフィルタとして、ファイル単位で処理を行うソートが挙げられる。

4　堆積可能フィルタ・ファイル・サーバの実現方式

堆積可能フィルタ・ファイル・サーバとは、フィルタ機能を持つ堆積可能ファイル・オブジェクトを扱うものである。提供するフィルタの性質により、異なる実現方式を提供する。本章では、まず、フィルタの性質によるサーバの分類を行う。次に、全てのサーバに共通な外部仕様と構成要素を示した後、個々のサーバの実現方式を示す。

4．1　サーバの分類

サーバは、以下の直交した2つの観点から分類される。

（1）フィルタの実現方式

サーバは、フィルタ処理をサーバ内部の手続きにより実現するものと、サーバ外部のプログラムを利用して実現するものに分けられる。

（2）一時オブジェクトの利用

一時オブジェクトとは、フィルタ処理の中間結果を一時的に保持するファイル・オブジェクトである。一時オブジェクトを利用するかどうかにより、サーバを分類することができる。一時オブジェクトを利用すると、行単位やファイル単位のフィルタを容易に実現できる。ソートの例では、サーバは、ソート処理後のデータ全体を一時オブジェクトに保存する。そして、データの読み出しを一時オブジェクトに対して行う。

以下の節において、フィルタの実現方式と一時オブジェクトの利用の違いによるサーバの実現方式を示す。4.4、4.5節では、一時オブジェクトを利用しないサーバの実現方式を示す。このサーバでは、行単位やファイル単位のフィルタ処理を扱わない。これにより、サーバの構成が単純化され、実現が容易となる。4.6節では、行単位やファイル単位のフィルタを、一時オブジェクトの利用により容易に実現するサーバを示す。

以下では、逆関数の存在するフィルタについて述べる。逆関数

のあるフィルタは、読み書き可能なファイル・オブジェクトを実現する。逆関数の存在しないフィルタは、書込み専用、または、読出し専用ファイル・オブジェクトを実現する。この時、フィルタは、読み書きのいずれかに設定される。例えば、読出し時に設定した場合、サーバは、書込み要求を受け付けないようにする。

4．2　ファイル・サーバの主要なサーバ手続きとオブジェクト手続き

2章で述べたように、オブジェクトの堆積では、一様なインタフェースを持つオブジェクトとサーバを用いる。以下に、堆積可能ファイル・サーバのインタフェースのうち、主要なサーバ手続きとオブジェクト手続きを示す。これらは、4.4節から4.6節で述べるサーバで共通に用いられる。

サーバ手続き

oid_t create_with_lower(oid_t lower[]; opt_t option[])

オブジェクト識別子の配列lowerで指定されたオブジェクトを下位層とする堆積オブジェクトを作成し、その識別子を返す。

オブジェクト手続き

attr_t object_get_attribute(oid_t oid)

オブジェクトの属性を返す。

void object_set_attribute(oid_t oid; attr_t attr)

オブジェクトの属性を変更する。

void_t object_kill(oid_t oid)

オブジェクトを消去をする。

buff_t file_read(oid_t oid; int where, len)

オブジェクトからデータの読出しを行う。引数として、読出す領域の先頭位置where、読出すデータの長さlenを与える。結果として、読出したデータを返す。

void file_write(oid_t oid; int where, len; buff_t buff)

オブジェクトへデータの書込みを行う。引数として、書込む領域の先頭位置where、書込む長さlenとデータbuffを与える。

4．3　サーバの構成要素

本節では、堆積可能フィルタ・ファイル・サーバを構成する4要素を示す。

（1）遠隔手続き呼出し受付け軽量プロセス

これは、クライアントからの遠隔手続き呼出しによる要求を受け付ける軽量プロセスである。この軽量プロセスは、普通、クライアントからの遠隔手続きを受け付ける状態にある。遠隔手続き呼出しを受け付けると、以下で述べる遠隔手続き実行軽量プロセスを生成する。

（2）遠隔手続き実行軽量プロセス

この軽量プロセスは、クライアントからの要求を並列に実行するためのものである。この軽量プロセスは、クライアントから1個にまとめて送られた遠隔手続きの引数を解き、手続き番号に対応するサーバ手続き、または、オブジェクト手続きを呼び出す。そして、遠隔手続きの結果を1個にまとめ、クライアントに返した後、終了する。

（3）サーバ手続き・オブジェクト手続き

この手続きは、遠隔手続き実行軽量プロセスにより呼出され、サーバやクライアントに指定されたオブジェクトへの操作を行うものである。サーバ手続きとオブジェクト手続きでは、アクセス件のチェックと並行性制御を行う。並行性制御として、1つのオブジェクトに対する複数の要求を並列に処理するため、軽量プロセス間の同期として実現されるロックによる相互排除を行う。

（4）オブジェクト保存部

オブジェクト保存部とは、サーバのメモリ中のオブジェクトをファイルへの保存や、ファイルからの読み出しを行うモジュールである。また、新たに堆積オブジェクトを生成する場合、オブジェクト識別子の割当てを行なう。ここで、オブジェクトを保存するファイルとは、下位層のオブジェクトではなく、サーバ自身のデータを保存するものである。

4．4　サーバ内部の手続きを利用するもの

本節は、一時オブジェクトを利用しない、サーバ内部の手続きによりフィルタを実現するサーバについて述べる。このサーバで扱えるフィルタは、バイト単位で処理を行うものに限られる。このサーバの動作を図2に示す。このサーバの特徴は、フィルタがサーバ内部の手続き呼出しで実現されるために、処理効率がよいことである。しかし、フィルタをサーバ内部で固定してしまうため、拡張性に乏しい。以下に、このサーバの処理の流れについて述べる。

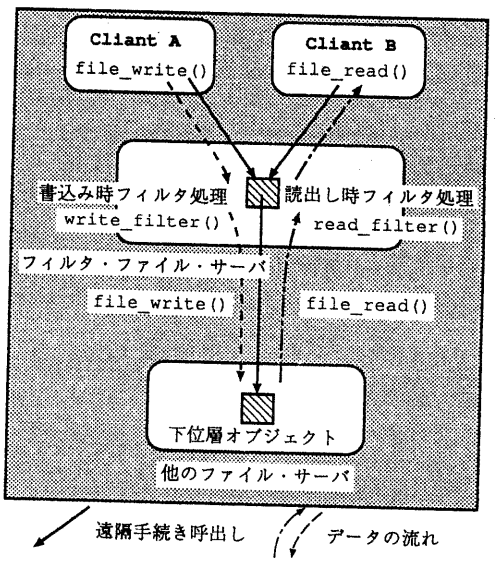

図2　サーバ内でフィルタ処理を実現するサーバ

サーバは、クライアントから書込み要求file_write()を受け付けると、受け取ったデータに対して、書き込み時のフィルタ処理をサーバ内手続きwrite_filter()により行う。そして、その結果を下位層オブジェクトへ書込むため、下位層のオブジェクトへfile_write()を呼び出す。

読出し要求file_read()を受け付けると、サーバは、下位層オブジェクトからクライアントにより指定された範囲のデータの読出しをfile_read()を呼び出して行う。そして、そのデータをサーバ内手続きread_filter()で読出し時フィルタ処理を行い、結果をクライアントに返す。

4．5　サーバ外部のプログラムを利用するサーバ

本節では、一時オブジェクトを利用しない、サーバ外部のプログラムを利用することでフィルタ処理を実現するサーバについて述べる（図3）。クライアントは、各オブジェクト作成時に書込み時と読出し時に行うフィルタ処理を行うプログラムを指定する。これにより、サーバは、複数のフィルタ処理を提供することが可能となる。このサーバで扱うことのできるプログラムは、データを入力ストリームから読み、処理を行い、結果を出力ストリームに書くものであり、クライアントによるアクセスの度に1つのプロセスとして実行される。サーバとのデータの入出力は、ストリームで行われ、サーバ内の軽量プロセスにより並列に処理される。

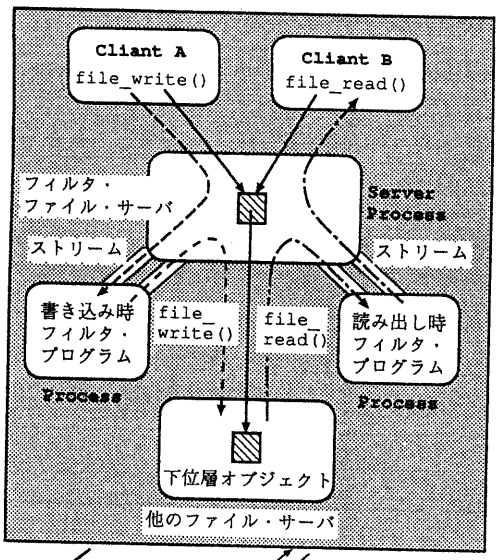

図3　サーバ外部のプログラムを利用するサーバ

4．6　一時オブジェクトを利用するサーバ

本節では、一時オブジェクトを利用して、バイト単位だけでなく、行単位、ファイル単位のフィルタ処理を実現するサーバについて述べる。このサーバの動作を図4に示す。このサーバの特徴は、フィルタ処理がクライアントのアクセスに関係なく、一時オブジェクトのデータ全体を下位層オブジェクトに書込む時と、下位層オブジェクトのデータ全体を読出し、一時オブジェクトへ書込む時にだけ行われる点である。これにより、サーバの処理効率が高められている。また、一時オブジェクトは、キャッシュとしての性格を合わせ持っているため、アクセスが高速化される。一時オブジェクトと下位層オブジェクト間のデータの一貫性制御を行うため、新たにキャッシングの方法を考案した。

下位層オブジェクトと一時オブジェクト間の一貫性制御を行う方法として、次の2つが挙げられる。1つは、クライアントからのアクセスの度に、一貫性の確認を行うものである。この方法では、強い一貫性が実現される。もう1つの方法は、クライアントからのアクセスとは無関係に、定期的に一貫性を確認するものである。この方法は、処理効率がよい。以下では、後者の方法を用いた一時オブジェクトの利用について述べる。

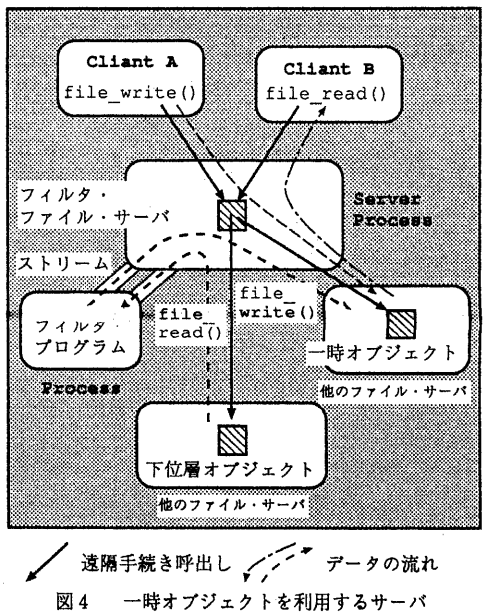

図4　一時オブジェクトを利用するサーバ

4．6．1　一時オブジェクトの状態

サーバは、一時オブジェクト利用のため、その識別子、属性と状態値を管理する。一時オブジェクトの状態には、一時オブジェクトの存在と、下位層オブジェクトのデータとの一貫性により、次の3状態を設定した。

INVALID ：　一時オブジェクトが存在していない状態。

VALID　：　一時オブジェクトが存在し、下位層オブジェクトとの間で一貫性がとれている状態。

DIRTY　：　一時オブジェクトが存在し、一時オブジェクトのデータが変更され、下位層オブジェクトとの間で一貫性がとれていない状態。

4．6．2　一時オブジェクトの状態遷移

一時オブジェクトの状態遷移図を図5に示す。サーバによる堆

積オブジェクトの作成、および、4.2節で述べたオブジェクト保存部によるファイルからの読出し時に、一時オブジェクトの状態は、初期値としてINVALIDに設定される。この状態でのクライアントによる読出し要求により、サーバは一時オブジェクトを作成し、状態はVALIDに遷移する。書込み要求の場合は、一時オブジェクトの作成後、データの変更が行われ、DIRTYに遷移する。サーバによる一時オブジェクトの変更の反映(copy-back)により、状態は、DIRTYからVALIDに遷移する。また、下位層のオブジェクトの変更の反映に伴う一時オブジェクトの消去(invalidate)により、状態は、INVALIDに遷移する。

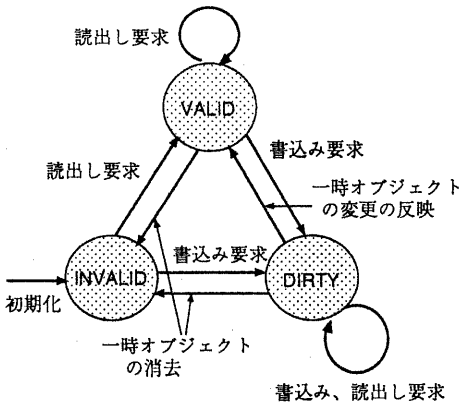

図5　一時オブジェクトの状態遷移図

4．6．3　一時オブジェクトへの操作

次に、サーバの一時オブジェクトに対する操作とその手順について述べる。

（1）一時オブジェクトの作成

サーバは、一時オブジェクトとなる新しいファイル・オブジェクトを作成するために、他のサーバに対してオブジェクト作成手続きを呼び出す。次に下位層オブジェクトからデータ全体を順次読み出し、そのデータについて読出し時フィルタ処理を行う。処理後のデータを作成したオブジェクトに書込む。

（2）一時オブジェクトの消去

サーバによる一時オブジェクトの消去は、次の２つの場合に行われる。１つは、クライアントによるオブジェクトへのアクセスがしばらく行われなくなった場合である。サーバは、状態がDIRTYの時、一時オブジェクトの消去の前にデータを書込み時フィルタ処理し、下位層オブジェクトへ書込むことで変更の反映を行う。もう１つは、下位層オブジェクトのデータが変更された場合である。この時、一時オブジェクトは、下位層に比べて古いため単純に消去される。

4．6．4　他のキャッシングの研究との比較

分散型ファイル・サーバの研究において、キャッシュと元データの間、あるいは、複製（replica）の間で一貫性を保つ効率的な方式を開発することは、重要な研究課題となっている［1］［2］［7］。これらの研究と比較して、ここで述べた方式の特徴は、以下のようになる。

（1）本方式の最大の特徴は、キャッシングの際に、フィルタ処理が行われることである。他の研究におけるでは、ファイルの内容は、単にコピーされるだけであり、ファイルの内容に対する処理は、行われない。

（2）読み書き可能なファイルについて、下位層のオブジェクトのデータと一時オブジェクトのデータは、対等である。これは、複製（replication）におけるコピー間の一貫性制御と類似している。ただし、本方式において、読込み専用の場合、下位層のオブジェクトの方が元データであり、書込み専用の場合は、一時オブジェクトの方が元データである。

（3）下位層のオブジェクトが変更されたことを、上位層のオブジェクトに伝える手段が提供されていない。これは、オブジェクトの堆積の制約により、下位層のオブジェクトは、上位層のオブジェクトの識別子を保持しないからである。

5　実現と性能

5．1　実現

４章で述べた実現方式に基づき、サーバを以下の環境で実現した。遠隔手続き呼出しには、SunRPC［3］を用いた。サーバ外部でフィルタを実現するにあたり、UNIXのコマンドの中でフィルタの機能を持つものを利用した。また、プロセスの生成は、UNIXのシステム・コールfork()とexecve()を用いた。サーバとフィルタ・プログラムを実行するプロセス間通信には、パイプを利用した。

軽量プロセスを実現するために、文献［5］で提案したマイクロプロセス・ライブラリを利用した。マイクロプロセス（利用者レベルの軽量プロセス）は、本来、カーネル・レベルの仮想プロセッサにより実行される。以下で述べる実験の結果は、１０個の仮想プロセッサを利用した。この数は、１つのサーバ内部における、同時に実行可能な入出力と遠隔手続き呼出しの数を決定する。以下の実験では、同一のサーバのオブジェクトを再帰的に積み重ねたので、そのために必要な最大の数の仮想プロセッサを生成した。

仮想プロセッサは、カーネルにおいて実現されるものである［5］。今回使用したライブラリでは、ＵＮＩＸのプロセスを用いてカーネル機能のエミュレーションを行っているため、ファイルの開閉（パイプの生成を含む）のオーバヘッドが非常に大きくなっている。すなわち、ソフトウェア割込みとプロセス間通信によるアクセス権の転送を用いているため、ファイルの開閉の操作の実行時間は、仮想プロセッサの数に線形に増加している。以下の実験においては、仮想プロセッサの数を１０に固定することで、このオーバヘッドを固定した。仮想プロセッサがカーネルにより実現されれば、このようなオーバヘッドは、生じない。

ここでは、４章で述べたサーバの実現方式に基づき、以下の３種のサーバを実現した。

サーバA．サーバ内部でフィルタを実現し、一時オブジェク

トを利用しないもの。

サーバB．サーバ外部のプログラムを利用してフィルタを実現し、一時オブジェクトを利用しないもの。

サーバC．サーバ外部のプログラムを利用してフィルタを実現し、一時オブジェクトを利用するもの。

さらに、実験では、主記憶を利用する基底ファイル・オブジェクトを提供するサーバを実現し、それを利用した。

5．2　性能

本節では、実現した各サーバの性能を測定する実験を行い、それらのサーバの性能を比較することでサーバの3種類の実現方式の特徴を示す。実験環境は、複数のワークステーションをイーサネットにより結合したものである。ワークステーションには、Sun4(SPARCstation ELC 33MHz) SunOS R4.1.2を使用した。基底ファイル・サーバと各堆積可能フィルタ・ファイル・サーバを別々の計算機上に配置した。実験では、クライアントが同一計算機にあるサーバに対してデータの読出し要求を行い、その処理時間を計測した。外部のプログラムを利用するサーバでは、書込み時と読出し時に、UNIXのフィルタ型コマンドcatを用いた。

5．2．1　フィルタの実現方式の違いによる性能比較

サーバAとBついて以下の実験を行い、性能の比較を行った。

（1）堆積回数と読出し実行時間の関係

条件：基底オブジェクトのデータの大きさ　1 Kbytes

クライアントが読出すデータの大きさ　1 Kbytes

実験結果を図6に示す。サーバAおよびBの両者において、堆積回数の増加につれ、読出し時間が線形に増加していることが分かる。1回の堆積における平均コストは、実現方式Aでは、約0.012秒、サーバBでは約0.40秒である。両者の差は、サーバ外部でフィルタ処理を行うプログラムの実行、パイプの作成、および、データの送受にかかる時間によるものである。

（2）読出すデータの大きさと読出し実行時間の関係

条件：基底オブジェクトのデータの大きさ　40 Kbytes

堆積の回数　1回

実験結果を図7に示す。サーバAおよびBの両者において、読出すデータ量が8Kbytseごとに読出し時間が階段状に増加する様子が認められた。この原因は、UDP/IPを用いたSunRPCにおける1回の遠隔手続きによるデータの最大量が8Kbytesであることによるものである。データのアクセス量がこの境界値(8Kbytes)を越えるごとに、遠隔手続き呼出しの回数と、サーバ外部のプログラムの実行回数が増加する。また、実現方式Bでは、境界における読出し時間の増加が(1)におけるAとBの差にほぼ一致する。これは、1回の遠隔手続き呼出しごとにサーバ外部でプログラムを実行するオーバヘッドが、遠隔手続き呼出しによる通信コストに比べて非常に大きいことを示している。

基底オブジェクトの大きさ 1 Kbytes
読出すデータの大きさ 1 Kbytes

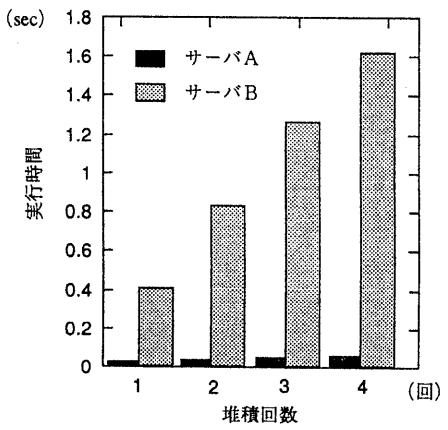

図6　堆積回数と読出し実行時間の関係

下位層オブジェクトの大きさ４０Ｋｂｙｔｅｓ
堆積回数　1 回

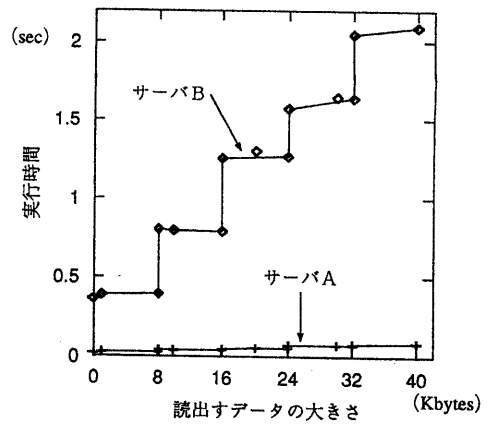

図7　読出すデータの大きさと読出し実行時間の関係

5．2．2　一時オブジェクト利用の有無による性能比較

サーバBとCについて以下の実験を行い、性能の比較を行った。

（1）堆積回数と読出し実行時間の関係

条件：基底オブジェクトのデータの大きさ　1 Kbytes

クライアントが読出すデータの大きさ　1 Kbytes

一時オブジェクトを利用するサーバCにおいては、測定を一時オブジェクトの存在する時(キャッシュのヒット時)としない時(キャッシュのミス時)についてそれぞれ行った。実験結果を図8に

基底オブジェクトの大きさ 1 Kbytes
読出すデータの大きさ 1 Kbytes

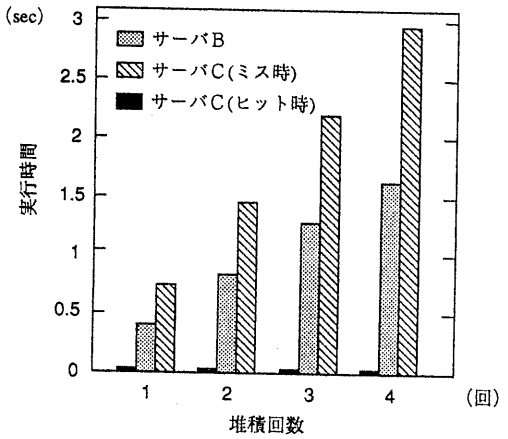

図8 堆積回数と読出し実行時間の関係

読出すのデータの大きさ 1 Kbytes
堆積回数 1 回

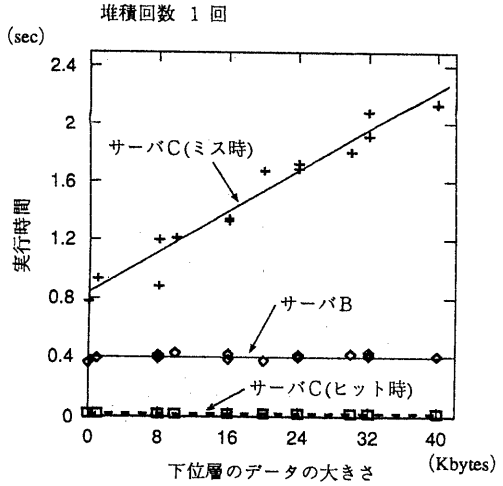

図9 下位層オブジェクトの大きさと読出し実行時間の関係

示す。サーバBの結果は、図6の実験結果と同じものである。サーバCのミス時における読出し時間は、サーバ外部でフィルタ処理を行うプログラムの実行時間に加えて、一時オブジェクトの作成時間がかかっている。堆積回数の増加につれて作成回数が増大することにより、読出し時間も大きくなる。サーバCのヒット時は、堆積の一番上に位置する一時オブジェクトからデータを読出すため、堆積回数に関係なく読出し時間が一定、かつ高速となる。これは、一時オブジェクトのキャッシュとしての効果を示している。

（2）下位層オブジェクトの大きさと読出し実行時間の関係
条件：クライアントが読出すデータの大きさ 1 Kbytes
堆積の回数 1回

実験結果を図9に示す。サーバBにおける読出し時間は、下位層のデータの大きさに関係なくほぼ一定となっている。サーバCのミス時には、下位層オブジェクトのデータ量が増加するごとに、読出し時間が大きくなっている。これは、一時オブジェクトの作成のために、サーバが下位層のデータ全体を読出すことに起因する。サーバCのヒット時には、キャッシュの効果によりサーバBに比較し、フィルタ処理を行わない分だけ読出し時間が高速になっている。

5．3 考察

これらの実験結果より、フィルタの実現において、サーバ外部のプログラムを実行するオーバヘッドが大きいことが分かった。また、一時オブジェクトによるキャッシュの効果が非常に大きいことが分かった。以上より、クライアントによる使用頻度の高いフィルタは、サーバ内部の手続きで実現するのが効率的である。使用頻度が低いフィルタの場合には、外部のプログラムの利用がサーバの拡張性の面で効率的である。

6 おわりに

本論文では、オブジェクトの堆積に基づくファイル・サーバの実現方式について述べた。オブジェクトの堆積は、object-basedの分散型オペレーティング・システムにおいて、複数のサーバが提供するオブジェクトの機能を統合して利用するためのモデル化の手法である。本論文で提案した実現方式の特徴は、外部のフィルタ機能を持つプログラムと一時オブジェクト（キャッシュ）を利用している点にあった。これにより、サーバの拡張性が高まり、性能が改善され、さらに、実現が容易になった。今後の課題としては、性能を改善することである。

参考文献

[1] R.G.Guy, J.S.Heidemann, Wai Mak, T.W.Page,Jr., G.J.Popek and D.Rothmeier: "Implementation of the Ficus replicated file system", USENIX 1990 Summer Conf., pp.63-71 (1990).

[2] 前川, 所, 清水 (編) : "分散型オペレーティングシステム", 共立出版 (1991).

[3] "Network Programming", Sun Microsystems,Inc. (1990).

[4] Y.Shinjo and Y.Kiyoki: "The Object-Stacking Model for Structuring Object-Based Systems", Proc. 2nd International Workshop on Object Orientation in Operating Systems (I-WOOOS'92), pp.328-340 (1992).

[5] 新城, 清木: "並列プログラムを対象とした軽量プロセス実現方式", 情報処理学会論文誌, Vol.33, No.1, pp.64-73 (1992).

[6] 新城, 清木: "分散型オペレーティング・システムにおけるオブジェクトの堆積", 情報処理学会第42回全国大会講演論文集（4）, 3K-8, pp.15-16 (1991).

[7] V.Srinivasan and J.C.Mogul: "Spritely NFS: Experiments with Cache-Consistency Protocols", SOSP12, ACM Operating System Review, Vol.23, No.5, pp.45-57 (1989).