システムソフトウェアと
オペレーティング・システム 65−22
（1994．7．22）

大域アドレス空間に基づく分散ＯＳにおける
オブジェクト移送の実現

古川 陽　　柴山悦哉
東京工業大学 大学院 理工学研究科 情報科学専攻

大域アドレス空間に基づく分散オペレーティングシステムGlassの概要と分散共有メモリサーバの実装について述べる．分散環境における大域アドレス空間ＯＳの構築のためには，効率的な分散共有メモリ技術が欠かせない．本稿では，まずアプリケーションプログラマの知識がオブジェクト移送のポリシーに十分反映されていないために生じる問題点を分析する．さらに，アプリケーションのプログラマが分散共有メモリの一貫性に関するヒントを提供すること，分散共有メモリサーバの開放的な実装を行うことにより，性能の改善が図れることを示した．

Object Migration in a Distributed Operating System based on the Global Address Space Model

Yo FURUKAWA　　Etsuya SHIBAYAMA
Department of Information Science,
Tokyo Institute of Technology

{furukawa,etsuya}@is.titech.ac.jp

This paper describes the basic concepts of the distributed operating system Glass, which is based on the global address space model, and design issues of its Distributed Shared Memory (DSM) server. An efficient implementation of DSM is essential in global address space operating systems. However, in distributed environments, the performance of DSM is usually poor due to the lack of application specific knowledge in the object migration policy. We show that exploiting the application specific knowledge and an open implementation of distributed shared memory server can significantly improve the DSM performance.

1 はじめに

並列計算機や分散システム向けの抽象化の方法として大域アドレス空間方式の有効性が認められている [2, 5]．大域アドレス空間オペレーティングシステムでは，単純であるが強力なプログラミングモデルと効率的な保護を両立することができる．

大域アドレス空間オペレーティングシステムの利用者が得る利点の多くは，スレッド間や保護ドメイン間でアドレス空間とメモリを共有できることに依っている．そのため，大域アドレス空間方式は共有メモリを介する通信との親和性がよいが，これを分散環境に応用する場合には，ソフトウェアで実現する分散共有メモリが性能低下の原因となる．

本稿では，分散共有メモリをメッセージ伝達方式と比較し，分散共有メモリのセマンティクスの問題点を検討する．アプリケーションと分散共有メモリサーバ間の知識交流が不充分であることが通信の効率化を妨げているという立場から，アプリケーションプログラマが分散共有メモリの一貫性に関するヒントを提供すること，分散共有メモリサーバの開放的な実装を行うことにより，性能の改善が図れることを示した．

現在，我々はPC/AT互換機上で動作するMach3.0マイクロカーネル上に大域アドレス空間オペレーティングシステムGlassを実装中である．本稿では，第2章でGlassの概要を述べた後，第3章で分散共有メモリの問題点を検討する．第4章では，その解決として分散共有メモリサーバの開放的な実装と，オブジェクト移送インタフェースについて解説する．第5章では，プロトタイプシステムによる実験の結果を示す．

2 大域アドレス空間オペレーティングシステム Glass

Glassは，大域アドレス空間とメモリオブジェクトによる抽象化を特徴とする分散オペレーティングシステムである．大域アドレス空間は分散環境におけるすべてのプロセスが単一のアドレス空間とマッピングを共有する方式である．また，メモリオブジェクトは記憶や計算結果を抽象化したオブジェクトである．大域アドレス空間とメモリオブジェクトにより単純なプログラミングモデルと効率的な保護機構を提供することができる．

この章ではGlassオペレーティングシステムの基本的な概念である大域アドレス空間，メモリオブジェクト，プロセスなどについて述べる．

2.1 大域アドレス空間

大域アドレス空間手法とは，オペレーティングシステムが管理するすべてのプロセスやオブジェクトが単一のアドレス空間を共有する方式である．そのため単一アドレス空間方式とも呼ばれる．オペレーティングシステムは管理するオブジェクトに大域的に一意なアドレス付けを行い，プロセスはその大域アドレスによりオブジェクトを指定する．

4.3BSDをはじめとする従来のオペレーティングシステムでは，プロセスが固有のアドレス空間を持っており，プロセスごとに異なるアドレス空間を切り替えて使用する．オペレーティングシステムはプロセスの固有アドレス空間の中で一意なアドレス付けを行う．その場合，プログラム中のアドレスはプロセス内でのみ意味を持ち，プロセス間のアドレスの受渡しは一般に意味をなさない．

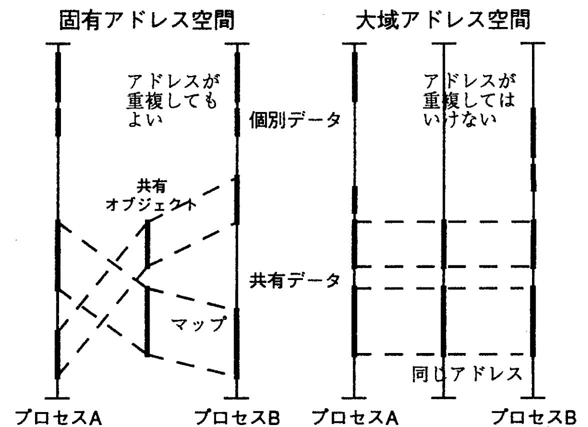


図1 固有アドレス空間と大域アドレス空間

大域アドレス空間手法はプロセス固有のアドレス空間と比較して次のような特徴がある．

- アドレス空間を大域的な名前空間として利用できる
  オペレーティングシステムが管理するすべてのオブジェクトはバイト単位でアドレス付けされる．この大域アドレスはシステム全体で一意な解釈を持つため，大域アドレス空間をフラットな名前空間として利用することができる．

- アドレスやデータ構造を共有できる
  プロセスごとに固有のアドレス空間をもつシステムでは，アドレスはプロセス内部でしか意味を持たない．一方，大域アドレス空間ではオブジェクトは(プロセス間でも)一意にアドレス付けされるため，アドレスはプロセス間で同じ意味を持つことができる．そのため，ポインタを含む複雑なデータ構造をそのまま共有することができる．一般に，リスト構造やツリー構造を含む複雑なデータ構造はポインタをいくつもたどって参照を行うが，プロセスごとに固有のアドレス空間をもつシステムではポインタが意味を持たず，間接的なポインタによるデータ構造の構築と参照を行わなくてはならない．

また，大域アドレス空間の導入によりデータだけでなくプログラムコードも共有しやすくなる．コードを含むメモリオブジェクトを共有することにより，共有ライブラリに相当する機構を容易に提供することができる．

2.2 メモリオブジェクト

Mach などにおけるメモリオブジェクトは記憶を抽象化したオブジェクトである．Glass においては，物理メモリなどの一次記憶，ディスク装置などの二次記憶に格納される記憶データ，プロセッサの計算結果などは，すべてメモリオブジェクトという形に抽象化され，ユーザはメモリオブジェクトをマップして利用する．

メモリオブジェクトは，生成時に大域アドレスにより名前付けされ，大域アドレス空間上の互いに交差しない範囲として管理される．4.3BSD などのファイルによる抽象化との相違のひとつに，シンボリックなファイル名による階層的な名前空間の概念とオブジェクトを分離したことがある．Glass ではメモリオブジェクトの先頭アドレスとサイズが管理され，大域アドレス空間がメモリオブジェクトのフラットな名前空間として機能する．メモリオブジェクトは，実装上は，ハードウェアで管理されるページの集まりである．

しかし，すべてのプロセスから任意のメモリオブジェクトが参照できるのでは，プロセス間のメモリオブジェクトの保護を行うことができない．このため，保護ドメインごとに大域アドレス空間の視野を制限する必要がある．具体的には，メモリオブジェクトのマップを制限するなどの方法によりアクセス制限を行う．

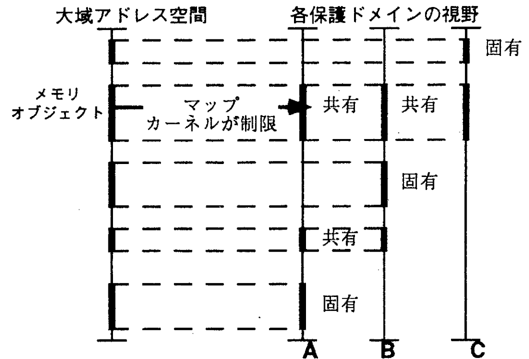


図 2 視野の制限

Glass のメモリオブジェクトは，プロセスの生存期間だけ有効な一時記憶としてもプロセスが消滅した後にも残る永続記憶としても利用される．永続記憶と一時記憶は統一的に扱われ，管理される．また，通常のファイルシステムに相当する機能は永続メモリオブジェクトと階層的なシンボリック名を提供するネームサーバ，認証サーバなどの組合せにより提供される．

2.3 プロセス

プロセスは，いくつかのスレッドから構成されるスレッドグループであるとともに，保護ドメインの役割を果たす．プロセスはメモリオブジェクト割り当ての単位であり，同じプロセスに所属するスレッドはメモリオブジェクトへのケイパビリティを共有する．つまり，スレッドがアクセスできるメモリオブジェクトは，所属するプロセスにより制限される．

スレッドはプロセッサでの計算を抽象化したオブジェクトであり，プログラムの論理的な実行主体である．スレッドはスタックやレジスタをはじめとする実行コンテクストを持つ．スケジューリングはスレッドを単位として行われる．

2.4 プロセスの展開と分散共有メモリ

プロセスは，複数のホストにまたがって展開することができる．スレッドを複数のホストに分散させることにより，並列処理やスレッド移送を実現することができる．

Glass では，仮想的にアドレス空間とメモリが共有されるので，スレッド移送時のコンテクスト回復

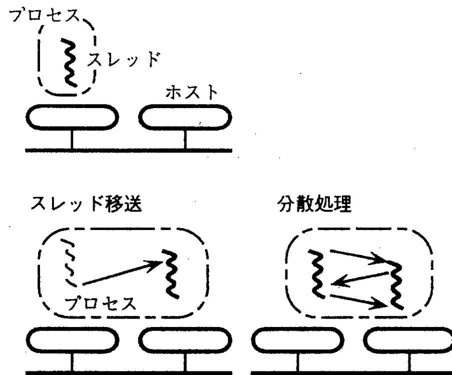


図 3 プロセスとスレッド

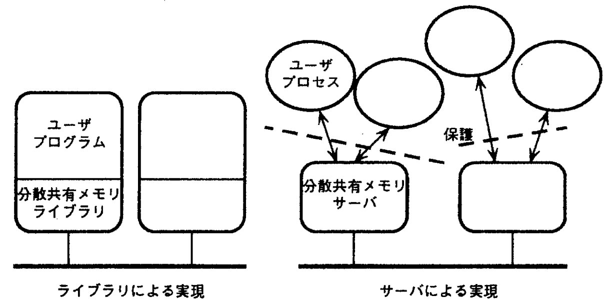


図 4 分散共有メモリの実現

が容易である．また，ホストをまたがるマルチスレッドの処理も従来のプロセス内のマルチスレッドプログラミングと同様に記述することができる．

3　分散共有メモリの問題点

オペレーティングシステムが提供するプロセス間通信機構として，メッセージ伝達方式方式と共有メモリ方式がある．前に述べたように，大域アドレス空間手法は共有メモリによるプロセス間通信との親和性が高い．複数のプロセスは，大域アドレス空間を共有しているので，同じメモリを同じアドレスによって読み書きすることにより，互いに通信することができる．しかし，分散環境においてソフトウェアで実現される共有メモリは現状としてあまり効率的ではない．この章では，効率低下の原因を検討する．

3.1　分散共有メモリの実現法

分散環境における共有メモリの実現法は，ハードウェアによるものとソフトウェアによるものに大別される．また，ソフトウェアによる実現は，ライブラリによるものと，サーバによるものに分けることができる．本稿では，オペレーティングシステムの基本機能として提供するために，サーバによる実現を検討する．

3.2　メッセージ伝達による通信と共有メモリによる通信の違い

メッセージ伝達方式では，プロセスからプロセスにメッセージを受け渡すことにより通信が行われる．

- **プログラマによる明示的な通信**
  通信のためには，送信側でメッセージを作る (pack)→メッセージを送る (send)，受信側でメッセージを受け取る (receive)→ メッセージをほどく (unpack) という一連の操作が必要となり，これらをプログラマがプログラム中に記述する．すべての通信がプログラマの記述にしたがって行われ，それ以外の通信は行われない．
- **同期点の存在**
  一般にメッセージ伝達では，送信側が `send` プリミティブの中で送ったメッセージを受信側が `receive` プリミティブの中で受け取るという合意がある．すなわち，プログラム中にデータの一貫性について同期する点が存在する．

一方，共有メモリでは，プロセス間で共有されるメモリを読み書きすることにより通信が行われる．

- **サーバによる暗黙的な通信**
  通常のメモリアクセス (load, store) のセマンティクスの中で通信が実現される．プログラムには共有メモリに対する読み書きのみを記述し，通信はサーバにより自動的におこわれる．そのため，共有メモリのセマンティクスの中には明示的な通信セッションの区切りがない．
- **サーバによる共有状態の管理**
  共有メモリにおいては，データの分散状況や共有状態をプログラマが特別に意識しなくても通信が可能である．共有データのアクセス時に通信相手を明示的に指定することはない．このような共有状態はシステムにより自動的に管理され，通信時にシステムが送信先を判断する．

- **同期点の不在**
  共有メモリのもうひとつの性質として，セマンティクスの中に同期を含まないことがあげられる．共有データに対する読み書きの逐次性は一般に保証されない．同期的なデータ共有や通信のためには，共有データをロックするなどの別の同期プリミティブを用いる必要がある．

3.3 効率低下の原因

分散共有メモリがメッセージ伝達方式によるものと比較して効率が悪くなる直接の原因として次のようなものが挙げられる．

1. **管理の単位が大きい**
   分散共有メモリでは，メモリアクセスの監視にはMMUなどのハードウェアを使うため，管理単位は一般にページ単位(典型的には4Kバイトなど)である．また，ページのサイズを小さくすると，サーバが共有状態を管理するために必要な管理情報が膨大となり現実的でない．そのため，本来独立に管理され転送されるべきデータが同じページ内に配置されるフォルスシェアリング(false sharing)になりやすい．
2. **無駄な通信がある**
   分散共有メモリでは，すべての通信や同期をシステムが自動的に行う．そのため，本来アプリケーションの実行には必要のない無駄な通信や同期が行われることがある．また，スラッシングなどの現象が起こることがある．
3. **転送のタイミング**
   分散共有メモリでは読み書きなどのアクセスが行われたときに，サーバの判断により通信が行われる．このとき，必要なデータやロックが得られるまでスレッドはブロックされる．すなわち，すべての通信は必要になったときに行われ，スレッドの実行とデータの転送を並列に行うことができない．

これらのうち，(1)の転送単位の問題はハードウェアの制限によるものであるので，ソフトウェアレベルでの解決は難しい．一方，(2)や(3)の問題は，データを利用するアプリケーションとデータを転送するサーバが分離されて，互いに十分な意志疎通が行われていないことに原因がある．すなわち，

- アプリケーションの知識とサーバの知識が分離されて，交流する手段が存在しないこと
- 相互に相手に指示する手段と権限が乏しいこと

が原因である．前節に述べたようにアプリケーションは有効なデータの存在するノードやデータの共有状態に関する情報を持っていない．また，データの通信を明示的に指定することはできない．一方，サーバはアプリケーションがいつデータを必要とするのか，アプリケーション同士がいつ同期を必要とするのかについての情報を持っていない．

4 アプリケーション情報を用いたページ移送

分散共有メモリの効率化(最適化)のためには，サーバとアプリケーションが相互に知識を交換して，それを利用することが重要である．この極端な例としては，分散共有メモリの実現をアプリケーションプログラマに開放し，アプリケーションに組み込んで使用することが考えられる．しかし，分散共有メモリをオペレーティングシステムの基本的な機能として提供するためには，ある程度の抽象化と統一的なインタフェースを規定し，一定の保護を行う必要がある．また，分散共有メモリの実現は手間がかかり，すべてのプログラマに同様の負担を強いるは好ましくない．ここでは，プログラマの必要に応じて，段階的に分散共有メモリの実現をカスタマイズできる機構を考える．

4.1 知識の分離

分散共有メモリを提供するサーバとそれを利用するアプリケーションを別のプロセスとして実装する場合，以下のように知識が分離する．

アプリケーション側は以下の知識を持つ．

- **共有データに対するアクセス**
  アプリケーションのプログラマは，ある共有データに対してプロセスが読み出しのみを行うのか，読み書き両方を行うのかを知っている．また，それらのアクセスがどのような順序とタイミングで行われるのかを把握している．
- **共有データに必要な一貫性**
  すべてのプロセスが共有するデータについて厳密な一貫性を必要としているわけではない．アプリケーションのプログラマは，共有データが，どの時点で，どのプロセス間で，一致していれば良いかを知っている．

一方，サーバ側は以下の知識を持っている．

- **共有ページの管理情報**
  分散共有メモリサーバは，個々のページデータについて，誰が有効なデータを持っているか，誰がロックを持っているか，データがどのようにどのプロセスにより共有されているかについての知識を持っている．
- **クライアントの管理情報**
  個々のクライアントについて，どのページに対してどのアクセスが許可されているか，持っているデータが有効であるかについての知識を持っている．

さらに，サーバは共有ページと管理情報を操作する権限を持っている．一般に，権限の譲渡は保護の破壊につながるため，ここでは，アプリケーションの知識をサーバに反映させる方法を検討する．

4.2 アプリケーションの知識の静的な反映

分散共有メモリでは，アクセスの局所性を高めるためにページのキャッシュを行う．サーバ内には，このキャッシュを管理するための一貫性プロトコルが実装される．最適な一貫性プロトコルはアプリケーションの要求するセマンティクスやアクセスパタンにより異なる．一貫性を緩めることで，メッセージの数を減らしたり，送出を遅らすことができる．前節で述べたように，アプリケーションのプログラマは，共有データに対するアクセスパタンの知識を持っている．Munin[1] などでは，あらかじめいくつかの一貫性プロトコルを用意し，この知識を利用して一貫性プロトコルを選択することにより通信を効率化している．ここでは，メモリオブジェクトのマップ時に，一貫性プロトコルを指定することにより実現する．

さらに，分散共有メモリの特殊な利用のために，アプリケーションのプログラマが専用の一貫性プロトコルの記述し，登録して利用することが考えられる．Mach などのマイクロカーネルでは，従来カーネル内で行われていた仮想記憶管理をユーザレベルに開放し，ページフォルト時の処理をユーザが記述できる機能がある．ユーザレベルで仮想記憶管理をするサーバを外部ページャと呼ぶ．一般に，分散共有メモリの実装はこの外部ページャの機能を用いて実現される．LucasFS[6] などでは，この機能の一部をユーザに開放し，ユーザがプロトコルをカスタマイズすることができる．Glass においても，分散共有メモリの一貫性プロトコルを記述するための外部ページャインタフェースを提供する．

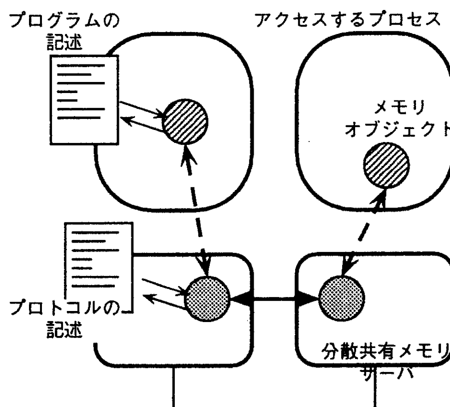


図 5 サーバとアプリケーションの関係

4.3 アプリケーションの知識の動的な反映

アクセスパタンなどの静的な情報に対して，共有データにいつアクセスするかなどの情報は実行時にしか得ることができない．この情報を実行時にアプリケーションからサーバに通知する機構を提供することにより，分散共有メモリの効率化を実現できる．例えば，ページ対するアクセスに先立って，サーバに必要なページを通知することによりプリフェッチを行うことができる．

ここでは，プログラム中に注釈としてページに対するアクセスや一貫性に関する情報を記述することにより，分散共有メモリを効率化する機構を提供する．アプリケーションのプログラマは，ページの転送，無効化や更新の指定をを明示的に記述することができる．この情報は実行時にメッセージとしてサーバに通知される．これは，サーバ側から見れば，アプリケーションの知識をヒントとして動的に収集していることになり，アプリケーション側から見れば，ユーザレベルで一貫性の管理をしていることに相当する．

実際には以下の注釈インタフェースが提供される．

- `fetch(addr, mode, when)`
  アドレスで指定するページをキャッシュに読み込む．大域アドレス空間システムでは大域アドレスによりページを一意に定めることができる．
  `mode` により読み込むキャッシュに対するアクセス権 (READ/WRITE) を指定する．例えば，書き込むことがあらかじめ分っているページに

対して，データと同時に排他ロックを要求することによりメッセージ数を節約できる．
when により読み込むタイミングを SYNC, ASYNC, LAZY から選択する．SYNC を指定した場合，読み込みは同期的に行われ，キャッシュに読み込まれるまで呼出しは戻ってこない．ASYNC の場合は，読み込みを要求するメッセージをサーバに送ったまま，読み込みを待たずにユーザの処理が続行される．LAZY は後述する遅延送出の指定である．

- yield(addr, when)
この指定により自分をサーバにより管理されている読み手集合から外すことができる．読み手集合を小さくすることにより，キャッシュに書き込む際の無効化や更新の要求を節約することができる．

- invalidate(addr, when)
読み手集合に対してページの無効化を要求する．

- update(addr, when)
読み手集合に対してページの更新を要求する．

- flush(addr)
遅延されているメッセージを送出する．

LAZY を指定することにより，flush が呼ばれるまで処理を遅らせることができる．release consistency などの緩い一貫性プロトコルでは，ある同期点のみで一貫性が保証され，同期点の間での一貫性を緩めることによりメッセージ数を節約している．Munin では同期プリミティブが呼び出されたときに一貫性のための処理を行うが，ここでは flush がそれに相当する．

注意すべき点は，ここで述べた注釈による処理はサーバの一貫性管理とは独立に行われることである．前節で述べたサーバの一貫性管理に厳密なプロトコルを選択した場合，ユーザがまったく注釈を加えなくても，一貫性のための処理はは自動的に行われる．逆に，サーバに緩い一貫性を指定した場合でも，ユーザが明示的に同期点を指定することにより，プログラムの特定部分のみに厳密な一貫性を提供することができる．また，fetch, yield などの非破壊的なプリミティブを過って挿入した場合でも，プログラムのセマンティクスに影響はない．

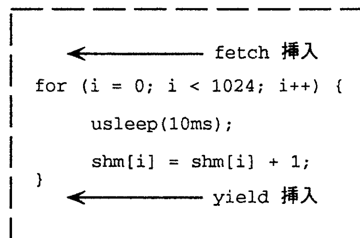
4.4 言語処理系の支援

前節で述べたような注釈を挿入する作業は，メッセージ伝達を用いてプログラミングする場合と比較したときには，大きな負担とはならない．しかし，必ずしも一般のプログラマが注釈を加えない場合でも，コンパイラがデータフロー解析などの結果を利用して自動的に挿入する方法なども考えることができる．

5 実験

本稿で述べた分散共有メモリサーバのプロトタイプシステムを PC/AT 互換機上で動作する BSD/386 上に実装し，実験を行った．各ホストは Ethernet で結合されている．メッセージ伝達のライブラリとして PVM を用いた．

- フォルスシェアリングの回避
1024 要素の整数配列に対して 2 つのプロセスが同時にアクセスする実験を行った．配列の各要素は同一ページ内の連続した領域に存在するため，アクセスの競合によりスラッシングが発生する．これは，fetch と yield の注釈を用いて，配列全体に対するアクセスを排他的なものにすることで，2 つのプロセスを逐次化し，アクセスの競合を避けることができる．この実験ではスラッシングが発生するようにループ内に 10ms の sleep を挿入し，各要素に対して読み込みと書き込みを行った．



	read fault	write fault	lock request	time [sec]
注釈なし	517	528	528	52.66
注釈あり	1	1	2	20.14

6 関連研究

Munin[1] は共有メモリ型マルチプロセッサ用に開発されたプログラムを分散環境で実行するために，分散共有メモリの最適化を行っている．アクセスパターンに応じた複数の一貫性プロトコルを用意し，アプリケーションのプログラマが細粒度の共有オブジェ

クトに対し，共有タイプを型として指定することにより選択する．Muninではあらかじめ用意されたプロトコルを用いるため，プログラマの負担はデータの型についての注釈にとどまるが，すべての用途に対して必要なプロトコルを用意できるわけではない．

LucasFS[6]はファイルシステムの一貫性制御のために，独自のプロトコル記述言語でユーザレベルで一貫性プロトコルを記述する機構を提供している．プロトコルはマップ時に指定する型情報により選択される．データへのアクセス情報以外には動的に情報を収集しないため，最適化の範囲も限定される．

CICO[4]はマルチプロセッサシステム上の共有メモリの効率化のために，`Check-in/Check-out`と呼ばれる注釈インタフェースを提供する．CICOはハードウェアの支援により厳密な一貫性を提供した上で，注釈を本稿に述べた`fetch`，`yield`に相当するプリミティブのように非破壊的なものに限定することで，プログラムのセマンティクスが変わらないことを保証している．その結果，データの一貫性に関してユーザレベルで制御することはできず，分散環境には必ずしも適さない．

7 まとめ

本稿では，大域アドレス空間手法を用いた分散オペレーティングシステムGlassの概要について述べた．さらに，分散環境における大域アドレス空間の構築のためには効率的な分散共有メモリ技術が不可欠であるという立場から，分散共有メモリの問題点を検討した．メッセージ伝達方式で一般に行われているように，分散共有メモリの効率化のためには，アプリケーションの知識を通信に十分反映することが必要である．本稿では，プロトコル記述の開放によるアプリケーションの知識の静的な反映と，プログラムに付加する注釈記述による動的に反映を検討した．本研究の成果をMach上に実装中のGlassに入れ，さらに実験を行う必要がある．

一般に，オペレーティングシステムの研究は，プログラマやユーザに対して資源の物理的な詳細を隠蔽し，抽象化して提供することに重きがおかれている．しかし，効率化や環境の動的変化に対応するためには，サーバとクライアントの密な交流が欠かせない．今後は，reflectiveなモデルを含めて，より柔軟な知識交流のモデルを検討し，課題とする．

謝辞

本研究を遂行するにあたり、文部省科学研究費重点領域研究「並列処理原理に基づく情報処理基本体系」(領域番号211)の研究グループ内における議論が、非常に有益であった。慶應義塾大学の斎藤信男教授をはじめとする同研究グループのメンバーに感謝いたします。

参考文献

[1] John B. Carter, John K. Bennett, and Willy Zwaenepoel. Implementation and Performance of Munin. In *Proceedings of 13th ACM Operating System Princeples*, pp. 152 – 164. ACM, 1991.

[2] Jeffrey S. Chase, Henry M. Levy, Edward D. Lazowska, and Miche Baker-Harvey. Lightweight Shared Objects in a 64-bit Operating System. In *Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA '92)*, pp. 397 – 413. ACM, 1992.

[3] P. Dasgupta, R. C. Chen, S. Menon, M. P. Pearson, R. Ananthanarayanan, U. Ramachandran, M. Ahamad, R. J. Leblanc, W. F. Appelbe, J. M. Bernabeu-Auban, P. W. Hutto, M. Y. A. Khalidi, and C. J. Wilkenloh. The Design and Implementation of the Clouds Distributed Operating System. Technical report, Collage of Computing Georgia Institute of Technology, Atlanta, Ga 30332-0280, 1990.

[4] Mark D. Hill, James R. Larus, Steven K. Reinhardt, and David A. Wood. Coöperative Shared Memory: Software and Hardware for Scalable Multiprocessors. *ACM Transactions on Computer Systems*, Vol. 11, No. 4, pp. 300 – 318, 1993.

[5] Michael L. Scott, Thomas J. LeBlanc, and Brian D. Marsh. Design Rationale for Psyche, a General-Purpose Multiprocessor Operating System. In *1988 ICPP*, pp. 255 – 262, 1988.

[6] 上原敬太郎, 宮沢元, 猪原茂和, 益田隆司. 分散協調作業のための一貫性プロトコルに基づく分散ファイルシステム. 情報処理学会研究報告書 94-OS-63, pp. 1 – 8. 情報処理学会, 1994.