システムソフトウェアと
オペレーティング・システム 61－6
（１９９３．８．１８）

Machの外部ページャによる分散共有メモリサーバとその応用

斎藤 彰一† 廣瀬 宰平† 大久保 英嗣† 大野 豊† 白川 洋充††

†立命館大学理工学部情報工学科

††近畿大学理工学部経営工学科

我々は，ネットワークワイドな共有メモリをユーザに提供する分散共有メモリサーバをMachオペレーティングシステム上に構築している．本サーバは，Machカーネルを変更することなくユーザレベルで実現し，タスクのアドレス空間管理，リモートマシンへのページングやネットワークワイドに共有する独自のスワップ領域の管理を行っている．

本サーバを用いて行った実験では，従来の１マシンでの処理速度を上回る結果を得た．また，本サーバを用いて作成された遠隔メソッド呼び出し(RPC)では，従来のRPCに必要とされたスタブを生成することなく，リモートマシンへの呼出しが可能となった．

An Implementation of Distributed Shared Memory Server by External Pager of Mach and Its Applications.

Shoichi Saito† Sakuhei Hirose† Eiji Okubo† Yutaka Ohno†
Hiromitsu Shirakawa††

†Department of Computer Science and Systems Engineering,
Faculty of Science and Engineering, Ritsumeikan University
56-1 Tojiin Kita-machi, Kita-ku, Kyoto 603, Japan

††Department of Industrial Engineering,
Faculty of Science and Engineering, Kinki University
3-4-1 Kowakae, Higashi-Osaka, 577, Japan

We are developing a distributed shared memory server on the Mach operating system, which provides users with a network-wide shared memory. This server is a user-level server and implemented without changing the Mach kernel. This server manages the address space of each task, the remote pagings, and the swapping storages which are shared with network sites.

Experimental results confirm that the performance of this server is better than that of a single machine case. Furthermore the remote procedure call by using this server can be performed in the same way as an ordinal procedure call. Namely it is not necessary to generate the stub procedures for the client and server.

1 はじめに

我々は, ネットワークワイドな共有メモリをユーザに提供する分散共有メモリサーバを Mach オペレーティングシステム [1] 上に構築している. 分散共有メモリ (Distributed Shared Memory)[2] は, ネットワークに配置された各タスクのアドレス空間の一部を互いに共有するための機構である. 我々が実装した分散共有メモリサーバは, 主記憶を分散共有メモリのキャッシュとして扱い, 各マシンにコピーを配置し, そのコピーに対して一貫性制御を行うことで, 各マシン上の共有メモリの内容の同一性を保っている. 本サーバを使用することによって, 共有メモリに対するアクセスは, 通常のメモリアクセスと同様に行うことが可能である. さらに, 本サーバでは, 分散共有メモリのためのスワップ領域をネットワーク全体で共通に使用できるように管理を行っている. スワップ領域をネットワークワイドに設定することの最大の利点は, ローカルのスワップ領域の大きさ以上のアドレス空間を使用できることにある. 即ち, 本サーバを使用することによって, 従来は大きな 2 次記憶を有するマシンでのみ可能であった処理が, 十分な大きさの 2 次記憶を持たないマシンにおいても可能となる.

以下, 本論文では, 我々が実装した分散共有メモリサーバの構成と一貫性制御方式について述べる. さらに, 本サーバを構成するページングマネージャと共有オブジェクトマネージャについて述べる. 最後に, オムロン製 LUNA88K(OS:UNIOS-MACH, Mach2.5) における性能評価と, 本サーバの応用例である遠隔メソッド呼出しについて述べる.

2 Mach の外部ページャ

分散共有メモリを実現するために, 我々はオペレーティングシステムとして Mach を採用した. Mach は, 従来のオペレーティングシステムではカーネルレベルでしか扱えなかった仮想記憶管理を, ユーザレベルで行なえる機能を提供している. これは外部ページャ(External Pager)[3][4] と呼ばれている. 外部ページャは, メモリオブジェクトに発生したページフォルトに対する動作を決定するタスクである. メモリオブジェクトは生成される時に特定のページャと結び付き, ページフォルトに対する動作が規定される. この機能はメモリオブジェクト単位で指定する. また, 1 つのマシンで複数の外部ページャを使用することも可能であり, メモリオブジェクト単位にページフォルトに対する動作を設定することができる.

Mach カーネルと外部ページャとの間のインタフェースを**外部メモリ管理インタフェース**という. この中で, 特にページの供給と削除に関するインタフェースを以下に示す. これは, 外部ページャがページイン/ページアウトを行うために使用するインタフェースであり, MIG(Mach Interface Generator) によって実装されている.

- **memory_object_data_request()**
  Mach カーネルから外部ページャに送られる RPC である. アクセスされるページが物理メモリ上にないときに, その要求として送られる.
- **memory_object_data_provided()**
  memory_object_data_request() で要求されたページを供給する. 外部ページャからカーネルへの RPC である.
- **memory_object_lock_request()**
  外部ページャが物理メモリにあるページのアクセス許可を変更する. また, ページの書き戻しをカーネルに要求する.
- **memory_object_data_write()**
  カーネルが書き戻しを要求されたページを外部ページャに送る.

3 分散共有メモリサーバ

分散共有メモリサーバはネットワークで接続されたマシン間でメモリオブジェクトの共有を行う. また, 2 次記憶の容量を超えるような処理を可能にするシステムである. これは, 各マシンに置かれた分散共有メモリサーバが協調して, マシンローカルなメモリオブジェクトを論理的に 1 つのメモリオブジェクトとして扱うことで実現している (図 1参照).

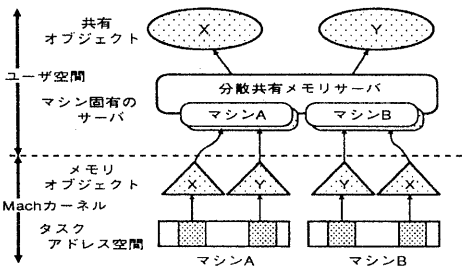

図 1 分散共有メモリサーバ

本サーバは外部ページャを利用した**分散共有メモリマネージャ(MM)** と, 共有メモリオブジェクトの管理を行う**共有オブジェクトマネージャ(OM)**, スワップ領域の管理を行う**ページングマネージャ(PM)** の 3 つのタスクで構成されている

(図2参照). 以下, それぞれについて説明する.

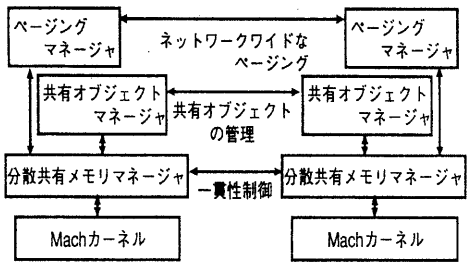

図 2　分散共有メモリサーバの構成

・**分散共有メモリマネージャ(MM)**

外部メモリ管理インタフェースを備えた外部ページャである. 主記憶上でページフォルトが発生した場合, カーネルからのメッセージに従ってメモリの供給と保存を行う. また, 他のサイトのメモリマネージャとの間で, メモリオブジェクトの一貫性制御を行う.

・**共有オブジェクトマネージャ(OM)**

共有オブジェクトの管理を行う. 本来メモリオブジェクトはマシンローカルなものである. 本マネージャでは, これを各サイト間で協調しながら管理を行ない, ネットワークワイドな共有オブジェクトとして実現している.

・**ページングマネージャ(PM)**

ページングマネージャは, ページング処理とそのための領域の管理を行なう. 本サーバのページング領域は, 各々のサイトが提供する領域をネットワーク上のすべてのサイトで共有している. 従来のページング領域はマシンローカルな存在であった. このために, 十分な2次記憶を確保できないマシンでは, ページング領域が不十分なために大きな仮想空間を利用するような処理を行えなかった. しかし, 本サーバではページング領域をすべてのサイトで共有することにより, その容量を超えるような大きな仮想空間を必要とする処理も行うことが可能となっている.

4　一貫性制御

分散共有メモリにおける一貫性制御は, ネットワーク上の複数のマシンに分散して存在するメモリイメージのコピーを, 読み出しや書き込みのアクセスに対してその内容を同一に保つ機構である. 本システムでは, 一貫性制御方式として**write-invalidate 方式**を使用している. この方式はあるサイトにおいて書き込みが起こると, その他のサイトが保有する同一ページのコピーを破棄することにより, ページの一貫性を保つ方法である. この方法では書き込みは同時に1つのスレッド, 読み出しは同時に複数のスレッドで行える (Single-WRITE Multiple-READ). 書き込みを行っているスレッドが属すマシンをそのページのページオーナという. ページオーナはページの最新の内容を保有するマシンを示し, そのページに対するアクセス要求の処理を行う. また, 本システムでは排他制御は一貫性制御の一環として行われる.

4.1　共有メモリページの状態

本システムでは共有メモリの基本単位として, 仮想記憶のページを用いている. 同時に一貫性制御の最小単位もページとなる. 我々が実装を行ったLUNA88K では1 ページは 8KB である. 共有メモリページの状態は**アクセス状態**によって表す. アクセス状態には, 複数のマシンが読み出す **Multiple-READ**, 唯1つのマシンが書き込む **Single-WRITE**, 排他制御機構によりページにロックを行う **LOCKED** の3つの状態がある.

ページはアクセス状態に対して**移行状態**を持つ. 移行状態とはあるアクセス状態に移行中の状態を表す. 移行状態は各アクセス状態にそれぞれ対応して存在する. これを**アクセス状態-shift** と呼んでいる. (Multiplr-READ 状態へ移行中ならば Multiple-READ-shift となる). よって, ページの状態は $3 * 2 = 6$ 通りで表される.

4.2　一貫性制御プロトコル

一貫性制御は MM によって行われる. クライアントタスクのアクセスによってページフォルトが発生した場合, カーネルは memory_object_data_request() 等の外部メモリ管理インタフェースを用いてローカル MM にページの要求を行う. ローカル MM はフォルトの発生したページのページオーナの検索 (後述) を行う. ページオーナが確定後, ローカル MM はページオーナ MM へ, ページのリモートリクエストを行う. これは, リモートリクエストをすべてページオーナに集めることで, ページ要求の逐次化を行うためである. これによって, ページフォルトに関する処理は, すべてページオーナ上で行うことが可能になる. 要求を受けたページオーナは, 要求が read 要求か write 要求かによって, **read fault** 処理か **write fault** 処理を行う. これらの処理はそのページの状態に基づいて場合分けされる. これらの処理中に行われることは, スワップ領域からのページイン, カーネルが物理メモリにマップし

ているページの書き戻し, ページオーナの変更などである.

4.3 ページオーナ検索

ページフォルトが発生した場合, サーバはページオーナにページの供給を要求する. このためページオーナがどのマシンにあるかを特定しなければならない. 以下ではページオーナの検索の方法について説明する.

すべてのマシンが最新のページの供給を受けるためには, 常に最新のページオーナを知っておくか, 必要になった時に素早くページオーナを検索できなければならない. これには以下の 3 つの方法が考えられる.

(1) ページオーナが変わる時に, ページオーナがブロードキャストを行い, ページオーナ以外のマシンの MM がこのメッセージを常に受けて最新のページオーナを知っておく方法.
(2) ページオーナが必要になるたびに, 必要になったマシンの MM がブロードキャストを行いページオーナを知る方法.
(3) ページオーナ管理用のマシンをあらかじめ設定しておき, ページオーナの変更をすべて把握しておく. ページオーナが必要になった時は, この管理マシンに問い合わせを行う方法.

本サーバは (3) の管理マシンを設定する方法で実装を行った. (1), (2) の方法には ブロードキャストが必要であるが, Mach の通信機構であるポートにはブロードキャストの機構がないためである. Mach のポートで同様のことを行うためには, すべての MM に個別にメッセージを送らなければならない. この方法ではマシンの数が増えた場合, ページオーナ検索に必要なメッセージ回数がマシン数の 2 乗に比例する形で増える (メッセージを送るマシン, 受けるマシンともに同数ずつ増えていくため). このために, 多数のマシンで構成するシステムにおいては, ページオーナ検索のためのコストがかかりすぎるので, 今回の実装では採用しなかった.

管理を行うマシンの負荷を分散させるために, 管理マシンは各共有オブジェクトごとに個別に設定する. ある共有オブジェクトをシステムの中で初めて使用したマシンを **home_machine** とし, これを管理マシンとしている.

4.4 排他制御

本システムにおける排他制御機構は MM へのユーザタスクからの IPC として実装されている. MM は LOCK を要求された共有ページを LOCKED 状態に移行させて, 他のタスクからのアクセスを禁止し, その共有ページに対して, write フォルト発生時と同様の処理を行うことでページを供給する. 即ち, 本システムの排他制御は, MM の一貫性制御機構が write-invalidate 方式を用いていることを利用して実現している. また, この要求は, MM において制御の逐次化を行っている. これにより, アクセス順序の逆転といった問題の発生を防いでいる.

5 共有オブジェクト管理

ユーザが分散共有メモリを使用する時は, OM に共有オブジェクトのマップを要求する.OM はその共有オブジェクトが既に宣言され使用されているものか, 初めて宣言されたものかを調べ, 共有データに登録を行う.

5.1 共有オブジェクト

共有オブジェクトとは各マシンに分散して存在するメモリオブジェクトを, MM によって一貫性制御を行うことで論理的に同一のメモリオブジェクトとしてみなしたメモリオブジェクトである. この共有オブジェクトは本システムの分散共有メモリの基本単位であり, 動的にその大きさを変更することはできない. また, ユーザによって決められる名前によって, ネットワークワイドにユニークに決まる存在である. ユーザはこの名前 (文字列) を共有オブジェクトの識別を行うために使用する. ユーザがメモリオブジェクトの共有を行う時は, 共有を行いたい共有オブジェクトと同一の名前を使用して確保を行う. サーバ側ではこの文字列を **object_number** に置き換えて, 識別の効率化をはかっている. この object_number は OM によってユニークに割り付けられる 32 ビットの整数である. サーバ内での共有オブジェクトの識別はすべて object_number によって行われる.

5.2 共有オブジェクトの管理データ

各マシンの OM 自体が使用する管理データも, 本サーバが提供する分散共有メモリ上に確保される. 共有メモリの使用は, 名前の検索などによる管理データの使用によって発生する OM 間のメッセージパッシングを抑制し, 管理データ使用の効率化をはかるためである.

この管理データには各 OM が共有して使用する情報を格納している. 1 つの共有オブジェクトごとに図 3 の構造体を使用している. `dsm_name` はユーザによって付けられた名前, `home_machine` は一貫性制御に使用される管理マシン, `dsm_size` は共有オブジェクトのページ数, `user_count` はその共有オブジェクトを使用しているユーザ数を

それぞれ表す.

```
struct shared_data {
    char    dsm_name[17];
    char    home_machine[9];
    int     dsm_size;
    int     user_count;
};
```

図 3　管理データ

5.3　共有オブジェクトの確保・解放

共有オブジェクトの確保と解放は以下のプロトコルに基づいて行われる (図 4および図 5参照).

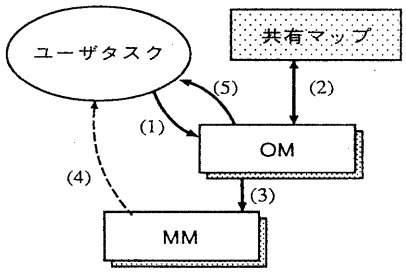

図 4　共有オブジェクトの確保

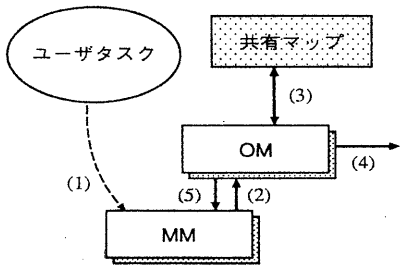

図 5　共有オブジェクトの解放

共有オブジェクトの確保

(1) ユーザが OM に共有オブジェクトの確保を要求する.
(2) OM は名前を基に現在使用されている共有オブジェクトを検索する. もし, 未使用ならば, 新しい共有データを作成する. また, 既に使用されている場合は, 該当する共有データの user_count の値を 1 つ増やす.
(3) OM は MM にメモリオブジェクトの生成を要求する.
(4) MM は内部データの初期化を行い, vm_map() を用いてユーザタスクにメモリオブジェクトをマップする.
(5) OM はユーザタスクに object_number を返す.

共有オブジェクトの解放

(1)　メモリオブジェクトの使用が終了すると, Mach カーネルから MM 内の memory_object_terminate() 関数が呼ばれる.
(2) この関数は直ちに OM 内の 終了処理関数を呼出して終了する.
(3) OM の終了処理関数は, 該当する共有データの user_count の値を 1 つ減らす. もしその値が 0 ならばすべてのユーザがその共有オブジェクトの使用を終了したことを意味するので, OM は共有データを消去する.
(4) すべての OM に対して共有オブジェクト用の内部データの解放を要求する.
(5) 各 OM は内部データの解放を行い, MM に対しても内部データの解放を要求する. MM の終了処理が終了した時点で共有オブジェクトの解放処理は終了する.

6　分散スワップ領域管理

外部ページャを利用することにより実現できる機能の 1 つにネットワークページングがある. これはページングをネットワーク上のリモートマシンとの間で行う機能である. 本システムではこの機能を利用することにより, ページングをネットワークワイドに行い (リモートページング), スワップ領域をマシン間で分散して確保・管理する機能を実現している.

スワップ領域を分散化することによる利点は, リモートマシンのスワップ領域を使用することで, ローカルマシン内のスワップ領域の大きさを超える大きさのスワップ領域を持つことを可能にする点である. これにより, スワップ領域以上の大きさの仮想記憶空間を利用することが可能になる. これは従来のディスクレスマシンのリモートマウントによるスワップ領域の確保とは異なり, すべてのマシンがスワップ領域を提供し合い共有することで, スワップ領域を介してメモリイメージの共有を可能にするものである. また, ネットワーク全体におけるスワップ領域の効率的な利用を行える.

本システムのスワップ領域は, UNIX ファイルとして通常の UNIX ファイルシステム内に存在している. このファイルはシステム起動時 (またはそれ以前) に確保され, 起動後にその大きさを変えることはできない. またスワップ領域は 1 共有メモリページ単位にアドレス付けされ, この単

位でページングが行われる.

6.1 リモートページング

リモートページングは, Mach の外部ページャを利用して実現している. リモートページングは, 従来はローカルなスワップ領域との間でのみ行われたページングを, 他のマシンのスワップ領域との間で行うものである. これは図 6 に示す構成で実現している. リモートマシンの 2 次記憶には直接アクセスできないので, またすべてのマシンのスワップ領域の状態を知ることは非常にコストがかかるので, リモートページングはローカル PM がリモート PM にページングを依頼する方法で行っている. そして, リモートページインは, 外部メモリ管理インタフェースが MIG で書かれていることから, リモート PM から直接ローカルのカーネルに供給する.

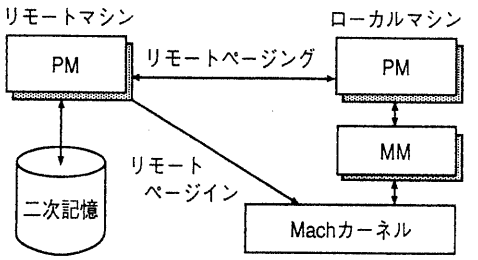

図 6 リモートページング

6.2 PM リンク

ローカルマシンのスワップ領域が溢れた場合, PM は他のマシンにリモートページングを依頼する. 依頼先の決定方法として以下に挙げる方法が考えられる.

(1) ブロードキャストを行い, それに最も早く反応したマシンにページングする.
(2) スワップ領域が最も空いているマシンにページングする.
(3) 依頼先のマシンをあらかじめ決めておく.

この 3 つの方法が考えられる. 本システムでは (3) の方法で実装を行っている. (1) のブロードキャストによる方法は 4.3節でも述べたように, Mach では非常にコストがかかる. また (2) の方法では, すべてのスワップ領域の利用率をすべてのマシンが把握する必要がある. (3) の方法では特定のマシンに依頼が集中しないようにしなければならない. そこで本システムではシステム全体の PM を巡回するリンクを構成し, 依頼先のマシンが分散するように実装を行った. また, ディスクレスマシンやリモートページングの依頼を受けるだけの 2 次記憶を持たないマシンは, クライアントマシンとしリンクの外に配置してある. 図 7 は, リンクを辿って, リモートページングを依頼する様子を示している. PM_a においてローカルスワップ領域が溢れた場合, リンクを辿って PM_b にリモートページングを依頼する. もし, PM_b でもローカルスワップ領域が一杯で依頼を受けられない場合は, 更に, PM_c に依頼が送られる. PM_c がページングを受けると, PM_a に了解のメッセージを送る. これにより, リモートページングが行われる. このようにクライアント PM はページングの依頼を受けることなしに, ページングの依頼のみを行う.

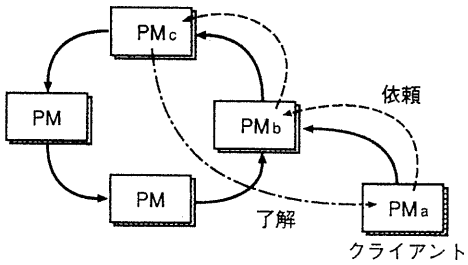

図 7 PM リンク

7 評価と応用

本章では, 簡単な例題によるシステムの性能評価と, 分散共有メモリを応用する時に問題となるネットワーク間でのページスラッシングについて述べる. さらに, 本システムの遠隔手続き呼出しへの応用についても述べる.

7.1 評価

評価の題材として行列積の計算を用いた. 共有メモリ上にマップされた 3 つの行列 A; B ;C に対し, C = AB の計算を行う. 各行列は異なる共有オブジェクトとして割り当てられる. 2 つのマシンに各 1 つの計算タスク, 1 つのマスタタスクを配置する. 以下に処理手順を示す.

(1) マスタタスクを起動し, 行列の生成と逆行列の計算を行う.
(2) 計算タスクを起動し, マスタタスクとの間にコネクションを張る.
(3) マスタタスクが行列を 2 分割し, 別々の計算タスクに割り当てる.
(4) 各々の計算が終了した時点で, マスタタスクが結果を検算して終了する.

行列の 2 分割は, 行列の次数を n とした場合, 0 列〜$\frac{n}{2}-1$ 列までと $\frac{n}{2}$ 列〜$n+1$ 列までで, 列を基

準に行う．測定結果を表1に示す．

表1　行列積の測定結果 (sec)

次数	32	64	128	256	512
分散処理	0.19	0.81	3.58	24.81	208.45
1タスク	0.05	0.53	4.10	38.96	366.66

この結果で示されるように，分散共有メモリを使用することによって，1タスクで得られる結果以上に良い結果を得ることができる．

7.2　集中アクセスとスラッシング

分散共有メモリの応用における問題点は，特定のアドレスへの集中的なアクセスとそれによって発生するスラッシングである．この場合のスラッシングとは，複数のマシンが1つのアドレスにアクセスを行い，これによりページがそれらのマシン間を激しく移動し続けるという，ページングの問題点をいう．本システムにおいてもこのスラッシングは発生する．複数のマシンが同時に同一ページに対して書き込みを行った場合である．本システムでは共有メモリの粒度が1ページ (8Kbytes) と大きいことから，数100bytes から数Kbytes 程度の共有メモリにおいて，スラッシングが発生する．

スラッシング問題の対応策として以下のことが考えられる．

(1) プログラミングによって集中アクセスを発生させない．
(2) 共有メモリの粒度を小さくし，スラッシングの発生による影響を少なくする．
(3) スレッドマイグレーションを行い，共有メモリへアクセスを行うスレッドを1つのマシンに集める．

それぞれについて，Mach 上での実現について検討する．(1) はユーザによって実現する方法と，新たな処理系によって行う方法がある．ユーザがプログラミングによって行う方法は Mach 上に限らず実現できるがユーザに大きな負担を強いることになる．また，新たな処理系の作成は簡単に行えるものではない．(2) の方法はページングを基にした本システムでは，仮想記憶のページサイズを変更することを意味する．これはカーネルやハードウェアに依存する部分であり，任意の大きさに変更はできない．また，外部ページャレベルで行うと，ページフォルトが発生したアドレスを，そのたびごとに検査しなければならないのでコストがかかりすぎる．(3) のスレッドマイグレーションは Mach ではサポートされておらず現実的ではない．このように3つの方法ともに問題点がある．

本システムを応用したシステムを作成する場合，(2) と (3) の方法は応用ごとにシステムの再設計を行う必要がある．また，(1) において新たな処理系の作成は行っていない．従って，現在はユーザのプログラミングにより集中アクセスの防止を行うという方法をとっている．これに関しては，今後の課題である．

7.3　遠隔メソッド呼出しへの応用

メッセージパッシングをベースとして遠隔手続き呼出し (RPC) を実現すると，手続き呼出しをメッセージに変換する操作が必要である．さらに，RPC のインタフェースの仕様をもとにスタブを生成しなければならないが，分散共有メモリを用いることでそれらの必要がなくなる．本節では，オブジェクト指向言語である Objective-C における遠隔メソッド呼出し (RMC) の共有メモリを使用した実装について述べる．

Objective-C ではメソッドはメッセージ式を記述することで実行される．このメッセージ式はメッセージャと呼ばれる一つの関数で実現されている (図8参照)．

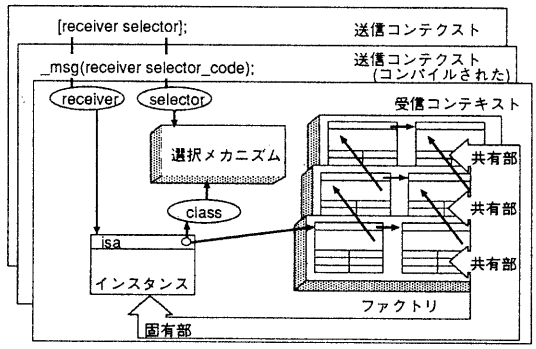

図8　Objective-C のメッセージャ

各クラスには共有部（ファクトリ）があり，メソッドのアドレスを動的に決定するためのディスパッチテーブルやインスタンスを生成するための情報が含まれる．メッセージャは固有部（インスタンス）の isa 変数からクラスのアドレスを得て，ファクトリを検索することでメソッドを決定する．

クライアントが用いるクラスをメソッドを実行するサーバとし，図8に示されるファクトリ群，インスタンスオブジェクト，またそれらを管理する情報を共有メモリにおくことで RMC を実現した．各クラスはクライアントと独立であり，メッセージ機構によりサーバの位置を動的に得るので，クライアントはサーバの位置を考慮することなしにプログラミングを行うことができる．本応用ではクラスとサーバを1対1としたが，複数の

クラスを1つのサーバとすることで，RMCを共有ライブラリとして用いることができる．以下，RMCの処理の流れについて述べる．

(1) 共有メモリの初期化マネジャを起動する．
(2) サーバは起動時に共有メモリにファクトリを登録し，クラス階層を構築する．このとき，メタクラスのファクトリはすでに共有メモリ上に存在する必要がある．
(3) クライアントは起動時に共有メモリ上のファクトリの位置を確認する．
(4) メッセージ要求が発生すると，クライアントはファクトリを検索しメソッドのアドレスおよびそのメソッドの所在するサーバのポートを得る．
(5) サーバに対し，メッセージを送る．これは同期をとるためだけのものである．
(6) サーバは要求のあったメソッドを実行し，返り値を共有メモリ上におき，メッセージを返す．

本応用の評価として，クラスを異なるホストで起動し，メソッドをサイト間で交互に呼出して，メソッド実行に要する時間を測定した(図9参照)．結果を表2に示す．

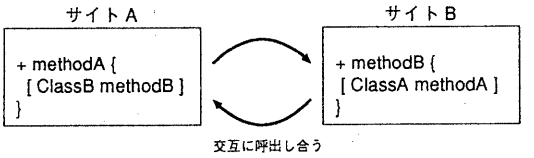

図9　メソッドの実行

表2　遠隔メソッド呼出しの測定結果(msec)

	実行に要する時間
2サイト2クラス	330.0
1サイト2クラス	7.0
スタブを用いたRPC	27.6

この結果で示されるように，サイト間でRMCを行うと1サイトに比べてかなり時間がかかる．これはメソッドを呼び出す側で書き込み，実行する側で同じページに対し読み出しが発生するので，スラッシングとほぼ同様の現象が生じるためである．

8　おわりに

本論文では，Machの外部ページャを用いた分散共有メモリサーバの構成とその評価，さらに応用について述べた．

本論文で述べた分散共有メモリサーバは，外部ページャを用いたユーザレベルにおける仮想記憶管理の実践例の1つである．応用例として示した行列積の計算のように，分散共有メモリを使用することで，分散処理の効率化をはかることができる．しかし，Machはスレッドがタスク間を移動することができないために，スラッシング問題に効率良く対応するシステムの作成は難しいといえる．現在は，この問題の対応策を検討している．

本システムはオムロン製 LUNA88K 上のMach2.5 において開発を行った．現在 GATEWAY 社製 GATEWAY2000 上の Mach3.0 で，本システムの Mach3.0 版を作成している．Mach3.0はマイクロカーネルとして実現されており，UNIXサーバなど多数の機能がユーザレベルで実現されている．これは我々が目指すユーザレベルでの分散システムの構築を，より容易にするものである．今後は，分散共有メモリを使用した各種アプリケーションの開発をマイクロカーネル Mach3.0 上で行う予定である．

参考文献

[1] Mike Accetta, Robert Baron, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young: *Mach:A New Kernel Foundation for UNIX Development*, 1986 Summer USENIX Conference (1986).

[2] Ming-Chit Tam, Jonathan M. Smith, and David J. Farber: *A Taxonomy-Based Comparison of Several Distributed Shared Memory Systems*, Operating System Review, Vol.24, No.3 (July,1990).

[3] Richard Rashid, Avadis Tevanian, Jr., Michael Young, David Golud, Robert Baron, David Black, William Bolosky, and Jonathan Chew: *Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architecture*, 2nd Architectural Support for Programming Languages and Operating System, ACM (1987).

[4] David Black: *External Memory Manager*, 1991 USENIX Mach symposium tutorial (1991).