分散永続性を提供するモーバイルオブジェクト・システムの実現

松原克弥† 加藤和彦†

近年，コンピュータネットワークの急速な普及にともない，多くのコンピュータがネットワークに接続され，情報の交換や共有が行われるようになってきた．このような分散処理環境での情報の交換や共有を目的とした分散協調処理アプリケーションの構築を支援するシステムとして，分散共有格納庫（Distributed Shared Repository: DSR）システムの開発を行っている．本システムの最大の特徴は，仮想記憶空間内のオブジェクトを仮想記憶空間から分離可能としたモーバイルオブジェクトの概念に基づいてシステム全体の設計が行われている点にある．仮想記憶空間から分離したモーバイルオブジェクトは，ネットワークで接続されたコンピュータの間を移動・巡回したり，システム内の永続記憶空間を用いて永続化することが可能である，本稿では，DSR システムを分散仮想記憶技術を用いて効率的に実現する方法について述べる．

Implementation of the Mobile Object System Providing Both Distribution and Persistency in a Uniform Way

KATSUYA MATSUBARA† and KAZUHIKO KATO†

The distributed shared repository (DSR for short) system is designed to develop distributed cooperative applications on the top of it. In the system, the uniform treatment of distribution and persistency is attained by seperating objects from a virtual address space. In the system, objects can move between different address spaces, and can be stored in persistent stores. We say that such objects are mobile. This paper describes an efficient implementation scheme of the DSR system with distributed virtual memory techniques. Some experimental results are shown to validate the design of the scheme.

1. はじめに

近年，計算機ネットワークは目覚ましい速度で広範な普及をとげており，今やワークステーションのみならず，安価なパーソナルコンピュータまでもがネットワークで結合されるようになった．これにともない，計算機処理の中心は，1台の大型汎用機を用いた集中処理から，ネットワークで接続された計算機群を用いた分散処理へと移行してきている．さらに最近は，ネットワークで接続された計算機間で情報の交換や共有を行うことを目的としたCSCW（Computer Supported Cooperative Work）やグループウェアなどの分散環境での協調処理を必要とするアプリケーションの開発がさかんに行われている．著者らは，このようなアプリケーションの構築を，プログラミング言語処理系に依存しないミドルウェアレベルから支援するシステムとして，分散共有格納庫システム（Distributed Shared Repository System，以下，DSR システム）の開発を行っている[7),10),15),16)]．

DSR システムの最大の特徴の1つは，計算の対象となるデータとそのデータへの操作を密閉したオブジェクトをネットワークで接続されたコンピュータサイト間を移動・巡回することにより，分散協調処理アプリケーションに必要な分散処理と永続処理を統一的な枠組みで実現できるようにしていることである．DSR システムが提供するシステムモデルは，仮想記憶空間上のオブジェクトを仮想記憶空間から分離して取りだし（この操作をアンロードと呼ぶ），それを DSR と呼ばれるコンピュータサイト間で共有された永続記憶空間に格納し，再びそれを別の仮想記憶空間に読み込むこと（この操作をロードと呼ぶ）を可能にしている．このように仮想記憶空間から分離可能とされたオブジェクトのことを**モーバイルオブジェクト**と呼ぶ．モーバイルオブジェクトは，内部に実行可能コード，データ領域（いわゆるヒープ領域を含む），そして CPU 実行状態（スタック領域と CPU レジスタを含む）を持つことができ，内部にスレッドを持たないパッシブオ

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

ブジェクトおよび内部にスレッドを持つアクティブオブジェクトの両方を表現可能である．

他のモーバイルオブジェクト・システムの多くは，バイトコードインタプリタ形式を用いてプログラミング言語処理系のレベルで実現を行っている．プログラミング言語処理系での実現では，特定のプログラミング言語（しばしば新たに設計されたプログラミング言語）を用いることが多く，プログラマにその言語の使用を強いる．また，これまでに蓄積されてきた様々なコンパイル時最適化技術を活用することが難しい．DSR システムでは，プログラミング言語の設計やその処理系の実現に対する制約を可能な限り少なくするとともに，実行時性能に関しても可能な限り犠牲にしないことを前提として設計が行われた．

以前に実装された DSR システムのプロトタイプシステム[7]では，モーバイルオブジェクトのロードおよびアンロード操作の際，モーバイルオブジェクトを単位として物理的なデータ転送を行っていた．このため，実質的にモーバイルオブジェクト中の一部のデータのみがロードされればよい場合でも，モーバイルオブジェクト全体を物理的にロードする必要があった．本稿では，DSR システムにおけるモーバイルオブジェクトのロードおよびアンロード操作に焦点を置き，効率的なロードおよびアンロード機構を実現する方法を述べる．

本実現では，仮想記憶管理技術の1つであるメモリマップ・ファイル機構を分散環境に拡張したリモートメモリマップ・ファイル機構を実現し，モーバイルオブジェクトのロードおよびアンロード時のデータ転送量を最少化する．さらに，内部に実行可能コードを含むオブジェクトをロードおよびアンロードを実現するために，アドレス空間依存情報（ポインタ）の反復的再配置技術を開発し，リモートメモリマップ・ファイル機構と調和的に統合する．また，内部にスレッドを持つオブジェクトのロードおよびアンロードを実現するために，スレッドの実行状態を取りだし，永続記憶空間に移動したり，別の仮想記憶空間上で継続する機能を実現する．

以下，本稿は次のように構成されている．2 章では，3 章以降の準備として，DSR システムの概要を簡潔に述べる．3 章では，分散環境に拡張したメモリマップ・ファイル機構について述べ，さらにその機構と反復的再配置技術を統合する一アプローチについて述べる．4 章では，3 章で述べた実現方法の UNIX 上での実装について述べる．5 章では，4 章で述べた実装システムを用いた実験について示す．6 章では，モーバイルオブジェクトとリモートメモリマップ・ファイル機構に関する関連研究について述べる．最後に 7 章で，まとめと今後の課題について述べる．

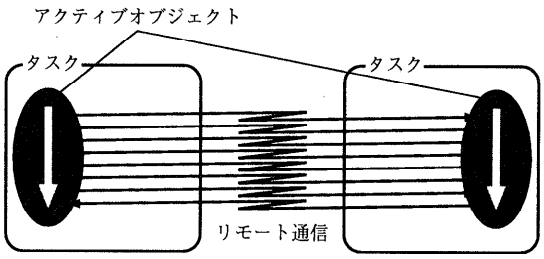


図1 メッセージパッシング
Fig. 1 Message passing.

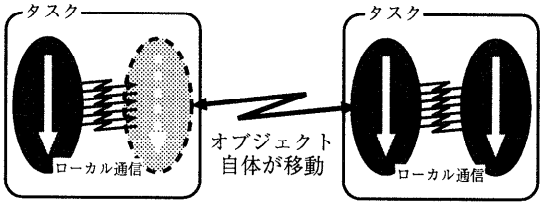


図2 オブジェクトパッシング
Fig. 2 Object passing.

2. 分散共有格納庫システム

DSR システムは，分散協調処理アプリケーションをプログラミング言語処理系に依存しないミドルウェアのレベルからサポートするシステムとして，以下のようなアプローチのもとで開発が行われている．

- **メッセージパッシングからオブジェクトパッシングへのパラダイムシフト**
  従来のプログラミングシステムの多くは，遠隔手続き呼び出し（RPC）などのメッセージパッシングを基本とするパラダイムに基づいている．地理的に離れたコンピュータサイト間で情報の交換や共有を行う分散協調処理では，サイト間の通信コストがシステム全体の性能に大きく関わる．このような通信のコストが多いアプリケーションでは，オブジェクト間のメッセージパッシング（**図1** 参照）から，オブジェクト自体が通信相手のコンピュータサイトに移動してローカル通信を行うオブジェクトパッシング（**図2** 参照）が有効であると考える．
  また，メッセージパッシングでは，コミュニケーションの際にデータのみが計算機間を移動し，データを扱うためのプログラムは各計算機であらかじめ別に用意する必要があった．不特定多数のユー

ザが動的に参加することが多いと予測される分散協調処理では，データとプログラムをカプセル化したオブジェクトをコミュニケーションの粒度とすることで，計算処理を自体を含む任意のデータをコミュニケーションに使用することが可能である．

- **オブジェクトに対する分散性と永続性の付与**
  電子メイルやネットワーク・ニュースシステムに代表されるような分散協調処理アプリケーションでは，分散処理に加えて情報に対する永続処理が必要となる．DSR システムでは，分散処理と永続処理をオブジェクトのモビリティを用いて統一的に扱い，それらをユーザ透明に実現することで分散性や永続性を意識することなくプログラミングが可能である．
- **ネイティブコードによるモーバイルオブジェクトの実現**
  現在提案されているモーバイルオブジェクト・システムの多くは，バイトコード・インタプリタなどの仮想機械上で動作するコードを用いてモーバイルオブジェクトを実現するアプローチをとっている．DSR システムでは，モーバイルオブジェクトの実行速度に注目し，ネイティブコードによってモーバイルオブジェクトを実現することで単一コンピュータサイトでの実行速度に匹敵する速度の実現を目指す．
- **プログラミング言語に「中立的」にモーバイルオブジェクトを実現**
  現在提案されているモーバイルオブジェクト・システムの多くは，モーバイルオブジェクトのための独自の言語を提案している．これに対し DSR システムは，プログラミング言語処理系に依存しないミドルウェアのレベルからモーバイルオブジェクトを実現する．DSR システムが定義するプログラミング言語処理系とのプロトコルに従えば，いかなる言語処理系も原理的に移植可能である．

2.1 基本概念

DSR システムのシステムモデルは，タスク，スレッド，モーバイルオブジェクト，DSR の 4 つの基本的な概念で説明される（**図 3** 参照）．

- タスク：線形的にアドレッシングが可能な仮想アドレス空間を指す．DSR 空間からモーバイルオブジェクトをロード，アンロードして計算を行う．
- モーバイルオブジェクト：DSR システムにおける情報管理の単位である．データ，プログラム，実行状態の 3 つを表現することができる．それぞれのモーバイルオブジェクトは，DSR システム内では一意で位置独立な名前を用いて識別する．
- スレッド：システムが提供する仮想プロセッサである．スレッドがモーバイルオブジェクトの上を走行することで実行が行われる．スレッドの数は実際のプロセッサの数に関係なく増減できる．
- DSR：すべてのタスクからアクセス可能な永続的な空間である．モーバイルオブジェクトは，DSR 空間に置かれることで永続性が付与される．タスクは，DSR を介してモーバイルオブジェクトの受渡しを行うことが可能である．DSR 空間にあるモーバイルオブジェクトの実際の格納は，物理的に離れたサイトに分散するすることが可能であるが，ユーザには透明になっており，この分散性を意識する必要はない．
  DSR の名前空間は，DSR を構成しているサイトの数や地理的な位置とは独立な木構造の単一名前空間を提供する．ロードするモーバイルオブジェクトの指定は，アンロード時に付けられた論理名を指定することで行う．指定された論理名は，システム内の分散名前解決機構によって，モーバイルオブジェクトが格納されている計算機サイト，ファイルシステム，およびそのファイルシステムでの名前が一意に特定される．タスク間でのモーバイルオブジェクトの移動は，すべて DSR を介して行う．送り側のタスクは，移動させるモーバイルオブジェクトを DSR にアンロードする．受取り側のタスクは，受け取るモーバイルオブジェクトの論理名を指定してロードプリミティブを呼び出す．ロードプリミティブは，DSR に指定した名前を持つモーバイルオブジェクトが DSR に現れるまでプリミティブ内で処理を一時中断する．

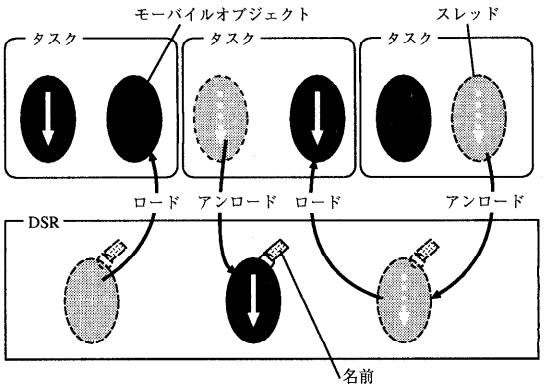


図 3 分散共有格納庫システムの基本概念
Fig. 3 Basic concepts of the DSR system.

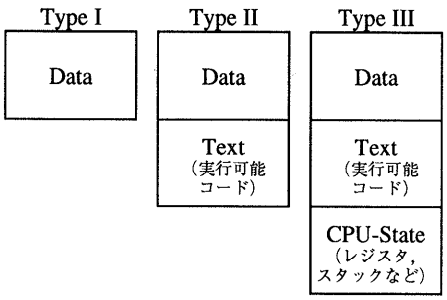


図4 モーバイルオブジェクトの種類
Fig. 4 Types of mobile objects.

指定した名前を持つモーバイルオブジェクトがDSRにアンロードされると，受取り側のタスクのブロックしていたプリミティブがモーバイルオブジェクトをタスクにロードして処理を再開する．

1つのモーバイルオブジェクトは，内部にいくつかのセグメントを持つ．セグメントには，データ，テキスト，CPU-Stateの3種類がある．データセグメントは，データが格納されているメモリ領域であり，書き込み可能である．テキストセグメントとは，実行可能コードが入るメモリ領域であり，プロセッサによって直接実行される．また，一般に書き込みは禁止されている．CPU-Stateセグメントは，スタックイメージ，プログラムカウンタやレジスタ値などが入るメモリ領域であり，書き込み可能になっている．これら3つを組み合わせることでモーバイルオブジェクトが形成される．DSRシステムで扱うモーバイルオブジェクトには3種類あり，セグメントの数に応じて，Type I，Type II，Type IIIと呼ぶ（図4参照）．Type Iモーバイルオブジェクトは，データセグメントのみを持つデータオブジェクトである．文書データ，ビットマップデータなどがこのモーバイルオブジェクトとして扱われる．Type IIモーバイルオブジェクトは，モーバイルオブジェクト内にスレッドが束縛されていないパッシブオブジェクトである．内部にテキストセグメントとデータセグメントを持ち，ライブラリなどの実行可能モジュールがこのタイプのオブジェクトで表現できる．Type IIIモーバイルオブジェクトは，モーバイルオブジェクト内にスレッドが束縛されていて，自らが能動的に活動するアクティブオブジェクトである．テキストセグメント，データセグメントに加えてCPU-Stateセグメントを持つ．このモーバイルオブジェクトは，プログラムの実行途中のイメージを表すのに用いられる．このType IIIモーバイルオブジェクトを用いて，計算過程のスナップショットをとったり，計算過程を別の計算機に移動させることができる．

2.2 基本プリミティブ

タスクは，dsr_in，dsr_out，dsr_read，dsr_writeの4つの基本プリミティブを発行することにより，モーバイルオブジェクトのロードおよびアンロードを指示する（図3参照）．

dsr_in（name，handle）は，名前nameを持つモーバイルオブジェクトをDSRからタスクにロードする．nameにマッチするモーバイルオブジェクトが見つかった場合，そのモーバイルオブジェクトは，dsr_inを発行したタスク上にロードされ，DSR上からは論理的に削除される．モーバイルオブジェクトのロード時には，モーバイルオブジェクト内部のアドレス参照の補正などの必要な処理が行われる．引数handleは，ロードした先頭アドレスなどのモーバイルオブジェクトに関する情報をシステムとタスクの間で交換するのに使われる．nameとマッチするモーバイルオブジェクトが見つからなかった場合には，そのプリミティブを発行したスレッドは，モーバイルオブジェクトが見つかるまでブロックされる．dsr_inプリミティブを用いることで，モーバイルオブジェクトの排他的ロード，およびモーバイルオブジェクトを使ったタスク間の同期を実現することが可能である．

dsr_read（name，handle）は，DSRからタスクにモーバイルオブジェクトをロードする点ではdsr_inと同じだが，ロードを行った後もDSR上からそのモーバイルオブジェクトは削除されない点で異なる．

dsr_out（name，handle）は，handleで指定されたモーバイルオブジェクトに名前nameをつけてタスクからDSRにアンロードする．アンロードされたコンテキストは，タスク上からは論理的に削除される．このnameは，DSR内でユニークでなければならない．

dsr_write（name，handle）は，アンロードされたタスク中のモーバイルオブジェクトが削除されないことを除いては，dsr_outと同じである．

本システムモデルの特徴的なアプローチは，分散処理と永続処理の両方をオブジェクトの移動の観点から統一的に扱うことである．つまり，分散処理を地理的に離れたサイト間のオブジェクトの移動にともなって行われる処理ととらえ，永続処理を揮発的な記憶空間と永続的な記憶空間との間のオブジェクトの移動にともなって行われる処理ととらえる[7]．DSRシステムモデルでは，前者をタスクからDSRを介した別のタスクへのモーバイルオブジェクトの移動，後者をタスクからDSRへのモーバイルオブジェクトのアンロードによって実現している．タスク間のモーバイルオブジェクトの移動は，DSRへのアンロード

操作（dsr_out, dsr_write）とタスクへのロード操作（dsr_in, dsr_read）を組み合わせて行う．

DSRプログラミングモデルを用いると，分散協調処理アプリケーションを構築するうえで必要となる様々な機能を系統的に実現することができる．文献7)では，異なるサイト上の2つのタスク間でオブジェクトを移動させるオブジェクトマイグレーションの例をあげている．

3. 分散仮想記憶技術を用いた実現

DSRシステムは，分散名前解決機構とロード/アンロード機構の2つから成る．分散名前解決機構とアクセス制御機構については，文献26)で報告を行った．本稿では，モーバイルオブジェクトのロード/アンロード機構実現について述べる．本実現法では，モーバイルオブジェクトのロードおよびアンロード操作を，分散仮想記憶技術を用いて効率的に行う．解決しなければならない技術課題を以下に示す．

(1) リモートメモリマップ・ファイル機構の分散透明な実現
(2) アドレス空間依存情報（ポインタ）の反復的再配置技術の確立
(3) 実行中のスレッドの動的なロードおよびアンロード技術の確立

第1のリモートメモリマップ・ファイル機構の実現により，メモリ管理ハードウェア（MMU）を利用してデータへのアクセス要求を効率的に検出し，必要最小限のデータ転送が可能になる．第2のアドレス空間依存情報の反復的再配置技術により，プログラムコードやアドレス参照データを含むType IIおよびType III モーバイルオブジェクトをタスクとDSRの間で繰返し的にロードおよびアンロード操作を行うことが可能になる．第3の実行中のスレッドの動的なロードおよびアンロード技術によって，内部にスレッドを持つType III モーバイルオブジェクトをタスクとDSRの間でロードおよびアンロード操作を行うことが可能になる．以下に，各々の技術課題に対する解決方法を述べる．

3.1 リモートメモリマップ・ファイル機構の実現

本節で述べるリモートメモリマップ・ファイル機構の実現法は，次の諸点に注意しながら設計を行った．

- **ポータビリティの重視** 従来，このような仮想記憶管理を操作するシステムを構築する際には，Machの外部ページャなどのユーザのレベルで仮想記憶管理部分をカスタマイズ可能なオペレーティングシステム上で実現されることが多い．本実現法は，外部ページャを必要とせず，最近設計されたオペレーティングシステムが提供している機能（仮想記憶の保護の設定機能，シグナルハンドリング機能）だけで実現可能である．
- **ローカルメモリマップとリモートメモリマップの透明性の保持** DSRシステムのシステムモデルでは，モーバイルオブジェクトが格納されている計算機の位置にかかわらず，分散透明にロードおよびアンロードが可能である．本実現法では，モーバイルオブジェクトが格納されている計算機の位置にかかわらず，ユーザが分散透明にモーバイルオブジェクトのメモリマップを行うことが可能である．
- **データ転送量の最少化** 異なる計算機に格納されているモーバイルオブジェクトをメモリマップする場合は，ネットワーク転送されたページをローカルディスクにキャッシュすることにより2度目以降の参照時のネットワーク転送を省くことが可能である．また，更新されたページ（dirty page）を管理することで，メモリアンマップ時のデータ転送量を最少化する．
- **commit/abort 機能の実現** 多くのオペレーティングシステムが提供するメモリマップ・ファイル機構では，ページの更新がただちにオリジナルのオブジェクトに反映されてしまうために，abortを行うことが難しい．本実現法では，オリジナルのモーバイルオブジェクトを直接メモリマップするのではなく，シャドウファイルをメモリマップすることでabort機能を実現する．この機能は，永続処理を行うシステムにおいては特に重要である．

本実現法のリモートメモリマップ・ファイル機構は，タスク内のタスク・スーパバイザ，アプリケーションが実行されているコンピュータサイトのローカルキャッシュ・マネージャとモーバイルオブジェクトのファイルが格納されているコンピュータサイトのファイルストア・マネージャの間での協調処理によって実現する．タスク・スーパバイザは，モーバイルオブジェクトをメモリマップしている仮想記憶領域で起こったページフォルトを捕えて，ページ要求，メモリマップの変更，dirty pageの記録を行う．ローカル・キャッシュマネージャは，リモートサイトから転送されてきたページのキャッシュ管理を行う．ファイルストア・マネージャは，ページ転送要求のあったページをモーバイルオブジェクトが格納されているファイルから読みだし転送する．また，転送されてきた更新ページをオリジナル

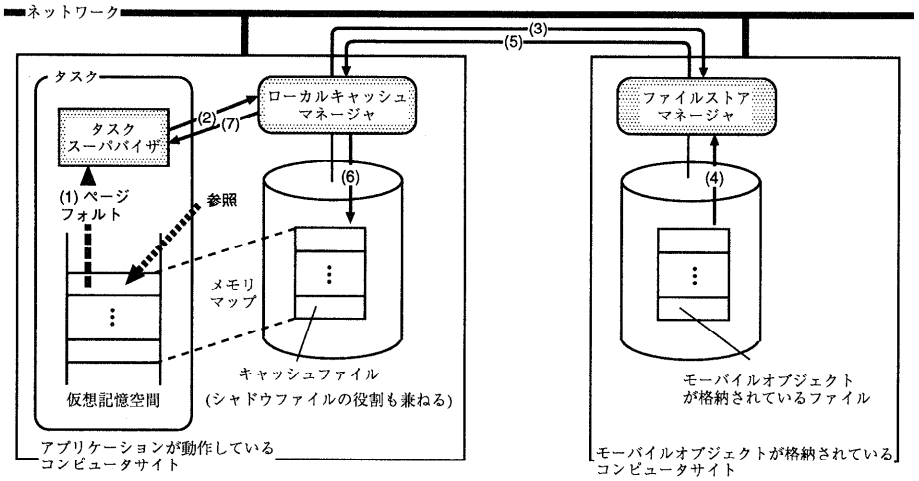


図5 リモートメモリマップ・ファイル機構（タスクとモーバイルオブジェクトが別のコンピュータサイトにある場合）

Fig. 5 Remote memory-mapped file mechanism (a task and a mobile object are located on distinct computer sites).

のモーバイルオブジェクトに反映させる処理を行う．

DSR からタスクへのモーバイルオブジェクトの転送は，以下のような手順で行う（**図5** 参照）．

(1) モーバイルオブジェクトをリモートメモリマップしている領域をユーザプログラムが参照すると，ページフォルトが発生してその命令がブロックされ，オペレーティングシステムのシグナル処理の機構によりタスク・スーパバイザが起動される．

(2) タスクスーパバイザは，ローカルキャッシュ・マネージャにページフォルトが起きたページをタスクの仮想記憶空間にページインするように要求する．

(3) ローカルキャッシュ・マネージャは，ページイン要求がマネージャが管理しているキャッシュファイルに存在するかを確認する．キャッシュファイルに存在していない場合には，そのページを含むモーバイルオブジェクトが格納されているコンピュータサイトのファイルストア・マネージャにページデータを転送するように要求する．

(4) ローカルキャッシュ・マネージャからの要求を受けたファイルストア・マネージャは，ファイルストアに格納されているモーバイルオブジェクトのファイルからページデータを読み込む．

(5) 読み込んだページデータを，要求してきたローカルキャッシュ・マネージャに転送する．

(6) ローカルキャッシュ・マネージャは，ファイルストア・マネージャから転送されてきたページデータを受け取り，タスクの仮想記憶空間にメモリマップしているキャッシュファイルにそのページデータを書き込む．

(7) 書き込みを終えたローカルキャッシュ・マネージャは，タスク・スーパバイザにページイン処理が終了したことを知らせるための確認メッセージを送る．

(8) タスク・スーパバイザは，ローカルキャッシュ・マネージャからの確認メッセージを受け取り，ページフォルトを起こしてブロックしていた命令を再開させる．

また，メモリマップするモーバイルオブジェクトがタスクと同じコンピュータサイトに存在する場合には，以下のような手順で copy-on-write のセマンティクスを実現する（**図6** 参照）．

メモリマップを行っている領域に書き込み不可のメモリ保護を設定しておき，書き込みが行われた際にのみページフォルトが起こるようにしておく．書き込みによるページフォルトが起こった場合は，その命令がブロックされてタスク・スーパバイザが起動される．タスク・スーパバイザは，書き込みが行われようとしたページをシャドウファイルにコピーして，そのページのメモリマップをシャドウファイルに変更する．その後，書き込み不可のメモリ保護を解除して，ページフォルトでブロックしていた命令を再開する．これにより，マップが変更されたページへの書き込みは，オリジナルのファイルではなくシャドウファイルに反映

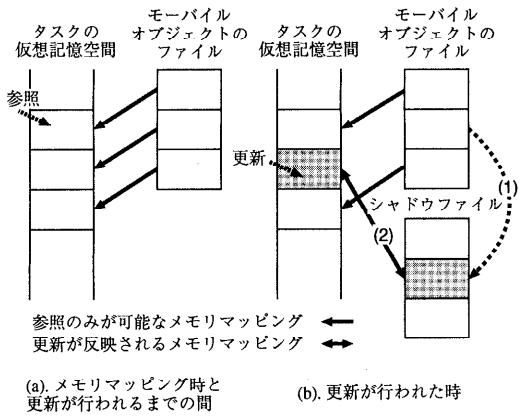


図6 リモートメモリマップ・ファイル機構（タスクとモーバイルオブジェクトが同じコンピュータサイトにある場合）

Fig. 6 Remote memory-mapped file mechanism (a task and a mobile object are located on the same computer site).

される．

実装システムでは，モーバイルオブジェクトのサイズが小さく，リモートメモリマップ・ファイル機構の利点が得られない場合の対策として，TCP/IP ストリーム通信を用いたオブジェクト単位の転送機構も実現している．本機構は，システム管理者によって設定された下限値パラメータに基づき，下限値パラメータよりも小さいオブジェクトの場合には，実行に先だってオブジェクト全体の転送を行う．リモートメモリマップ・ファイル機構の実行時の仮想記憶ページ単位の転送方法とオブジェクト単位の実行前の転送方法の選択は，ユーザ透明に行われる．

3.2 アドレス空間依存情報の反復的再配置の実現

DSR システムのシステムモデルでは，1つの仮想記憶空間（タスク）の任意の位置にモーバイルオブジェクトをロードすることができる．また，アンロードしたモーバイルオブジェクトを再び任意のタスクの任意の位置に再びロードできる．この機能を実現するために，モーバイルオブジェクト内のポインタなどのアドレス空間に依存した情報を繰り返し再配置する技術（反復的再配置技術）を実現する．以下に，モーバイルオブジェクト内の実行可能コードが含まれるテキストセグメントとポインタなどのデータ構造が含まれるデータセグメントにおける反復的再配置技術の実現方法を述べる．

3.2.1 テキストセグメントの反復的再配置

テキストセグメントの再配置とは，テキスト中のアドレスに依存した情報を，メモリマップした仮想記憶領域内に適合するように調整することを指す．この機構を実現するために，再配置可能コードの技術[2]を用いる．再配置可能コードは，実行時に特別な処理を必要としないために，繰り返しコードが再配置される反復的再配置と親和性が高い．再配置可能コードは，プログラムコード中の call 命令や jump 命令に用いるアドレスをレジスタを用いた間接アドレッシングによって記述する．load 命令や store 命令で指定するアドレスは，データのアドレス指定用のテーブルを用意して，テーブルを介した間接アクセスを使って指定する．

3.2.2 データセグメントの反復的再配置

データセグメントの再配置とは，データ構造へのポインタなどのアドレス空間に依存した情報を，新たにメモリマップした仮想記憶領域内に適合するように調整することを指す．データセグメントは，コンパイル時に静的に割り当てられる領域と実行時に動的に割り当てられる領域に分けられる．以下に，各々の領域に対する実現方法を述べる．

データセグメントの再配置を実現するには，セグメント内のアドレス空間依存情報の位置の管理が必要になる．静的に割り当てられるデータ領域は，コンパイル時に得られる変数の型情報から位置情報を得る．実行中に動的に割り当てられるデータ領域は，Pascal などの型制約の強いプログラミング言語の場合にはコンパイラにより得られる型情報を用いる．C などの型制約の弱いプログラミング言語の場合には，アプリケーションプログラムが実行時に型情報を明に指定する方法を用いる．

データセグメントの再配置は，アドレス空間依存情報の内容を直接変更する必要がある．モーバイルオブジェクトのロード時にオブジェクト内のすべてのアドレス空間依存情報の再配置を行うためには，オブジェクト全体を転送する必要がある．本実現法では，リモートメモリマップ・ファイル機構とアドレス空間依存情報の反復的再配置を調和的に統合するために，実行時にページ単位で再配置を行う．ページ単位で再配置を行う場合の技術的課題は，更新されたページと更新されていないページのアドレス参照に一貫性がなくなることである．この問題に対処する方策として，以下の3つが考えられる．

(1) **正準開始番地法** モーバイルオブジェクトが DSR 上にあるときは，モーバイルオブジェクト内のアドレス参照情報はすべてつねに正準開始番地からのオフセットとして表現される．更新されたページを DSR に書き戻すときは，書き戻すページ内のすべてのアドレス参照情報を正準開始番地からのオフセットの表現になおし

てからDSRに格納する．

(2) **モーバイルオブジェクト毎正準開始番地法** 正準開始番地法では，つねに一定のアドレスを正準開始番地としたが，この方法では，モーバイルオブジェクトを単位として正準開始番地を設定する．モーバイルオブジェクトをロードするときに，そのモーバイルオブジェクトの正準開始番地から始まるアドレス領域にすでに他のモーバイルオブジェクトがロードされていてロード不可能な場合に，ロード可能な開始アドレスを新たに選択し，その開始アドレスをそのモーバイルオブジェクトの新たな正準開始番地として設定する．ロードされたモーバイルオブジェクトがDSRに書き戻されるときには，モーバイルオブジェクト内のすべてのアドレス参照情報が，新たに正準開始番地からのオフセットになるように設定する．

(3) **ページ単位オフセット法** 正準開始番地を設定せず，ページを単位として再配置を行った際の情報をオブジェクトとともに記録する☆．ページをロードするたびに，そのページに関して記録した再配置情報を基に再配置を行う．モーバイルオブジェクト中のページは，それぞれ最も最近メモリマップを行った仮想記憶領域に適合するように再配置されたアドレス表現でDSRに格納される．アンロードされた後，再びロードするときは，ページ管理テーブルから前回の再配置情報を取得して新しくメモリマップを行った領域に適合するように再配置を行う．

第1の正準開始番地法は，オブジェクト内のアドレス空間依存情報がつねに仮想記憶空間の開始番地からのオフセットになっているので，メモリマップを行うときに複雑なアドレス計算を必要とせず，メモリマップ開始アドレスを加算する処理で再配置が可能である．しかし，書き戻し時に再配置を行ったアドレス空間依存情報をメモリマップ前の状態に戻す処理が必要となるため，メモリアンマップ時に一定のオーバヘッドが生じる．第2のモーバイルオブジェクト毎正準開始番地法は，第1の正準開始番地法で起きていたメモリアンマップ時の一定のオーバヘッドがない．しかし，オブジェクト作成時に決定した仮想記憶領域にメモリマップが行えない場合にオブジェクト全体に対する再配置処理が必要となるため，オブジェクト作成時に決定した仮想記憶領域にメモリマップを行うことができなかったときにオーバヘッドが生じる．第3のページ単位オフセット法は，再配置処理のオーバヘッドを遅延し，前回メモリマップを行った仮想記憶領域と異なる領域にメモリマップを行ったときのみページ単位で再配置処理を行う．しかし，再配置情報を記録するページごとの管理テーブルをオブジェクトに付加するため，メモリマップするオブジェクトのサイズの増加によるオーバヘッドが考えられる．ページ管理テーブルの大きさは，32ビットCPUの仮想記憶空間で4バイトで，1ページが4キロバイトとすると，1024ページ分の管理テーブルが1ページ追加するだけで入る．仮想アドレス空間は，モーバイルオブジェクトの一般的な大きさから比べると広大であり，疎に使用することができる．このため，モーバイルオブジェクトのメモリマップ時のアドレス領域の衝突を少なくすることが可能であるため，第1の正準開始番地法よりも，第2のモーバイルオブジェクト毎正準開始番地法または第3のページ単位オフセット法がより有効な方式であると考えられる．また，たかだか数ページの管理テーブルの追加で各ページに対する再配置を遅延させることができるため，第2のモーバイルオブジェクト毎正準開始番地法よりも第3のページ単位オフセット法が効率的と判断できる．以上の考察の結果，第3のページ単位オフセット法が最も効果的な管理方法であると判断し，本実現法に用いた．

オブジェクトのポインタの整合性を調整するための技術として，pointer swizzling/unswizzlingが広く知られている[1),24)]．pointer swizzling/unswizzling技術は，主にオブジェクト間の参照をアドレス空間に依存しない形式に変換することで，参照先のオブジェクトが別の場所に移動した場合でも，オブジェクトの位置に依存せずにオブジェクト参照を表現することを可能にする方法である．本実現法で提案するデータセグメントの再配置法は，仮想記憶空間内でオブジェクトの位置が変化したときに，オブジェクト内を参照するポインタの整合性を維持するための調整機構である．

3.3 スレッドのロード/アンロード機構の実現

2.1節で述べたように，DSRシステムのシステムモデルでは，モーバイルオブジェクト内に持つスレッドをCPU-Stateセグメントによって表現する．CPU-Stateセグメントは，モーバイルオブジェクト内のスレッドの現在の実行状態や実行環境などの情報を含む．スレッドのロードおよびアンロードとは，CPU-Stateセグメントをロードおよびアンロードすることを指す．Type IIIモーバイルオブジェクトのロード時に

☆ ページを粒度として正準開始番地を設定していると解釈することも可能である．

は，CPU-State セグメントを新たにメモリマップした仮想記憶領域内に適合するように調整する技術が必要となる．以下に，CPU-State セグメントのロードおよびアンロードの実現方法を述べる．

オブジェクトの計算状態の移動を実現する方法として，オブジェクトの状態が反映されているデータを移動させる方法がある．プログラムコード部分は，あらかじめシステム内の各計算機サイトで利用できるようにしておき，計算状態を表すデータ（数値計算における計算結果など）を，メッセージパッシングなどを機構を用いて転送することにより，転送先でオブジェクトの実行状態を再現する．この方法は，移動する時点でデータに計算状態が正しく反映されるように，アプリケーション全体もしくはプログラミング言語処理系を注意深く設計する必要がある．本稿で提案する実現法では，アプリケーションやプログラミング言語には透明に計算状態を移動することが可能で，原理的に任意の時点で移動することができる．

CPU-State セグメントには，スレッドのスタック，プログラムカウンタ，レジスタなどのスレッドが実行するのに必要な最低限の情報を格納する．スタック，プログラムカウンタとレジスタは，内部にアドレス空間依存情報を含む場合がある．これらに含まれるアドレス空間依存情報は，その位置や数が実行状態に応じて動的に変化する．本実現法では，スレッドを持つモーバイルオブジェクトをアンロードするときは，オブジェクト自身が明にアンロード・プリミティブを発行するという条件をシステムモデルに付加する．アンロード・プリミティブは，現在のスレッドの実行状態がスタックに退避されるのを利用してアドレス空間依存情報の位置を確定することができる．スタック内のアドレス空間依存情報の位置は，スタックフレームを解析して得られた情報とコンパイラが出力するシンボルテーブルの情報を用いることで得る[6),7)]．プログラムカウンタとレジスタは，アンロード・プリミティブを発行した時点でスタックに退避されるので，スタックと同様の方法でアドレス空間依存情報の位置を得ることができる．このようにして得られるアドレス空間依存情報に，データセグメントと同様のページ管理テーブルを用いた反復的再配置処理を行う．

本実現法と同様のユーザ透明な計算状態の移動を実現する方法として，アプリケーションを構成するプロセスの仮想記憶空間全体の情報をファイルにダンプして移動先に転送する方法がある．前者の方法に比べて，ユーザやアプリケーションに対する透明性が高く，原理的に任意の時点で移動することが可能である．それに対して本システムで実現しているのは，オブジェクトを粒度として仮想記憶空間から取り出す技術である．

スレッドのロード/アンロードにおける本実現法での最大の特徴は，プログラミング言語処理系に依存しないレベルでのスレッド状態をロード/アンロードを可能にしている点である．従来のシステムの多くは，状態を移動可能な場所をあらかじめ静的に決めておき，コンパイル後のバイナリコードに依存しないレベルでプログラムの実行状態を静的に生成する方法を用いている．本実現法は，命令コードレベルでの計算状態の移動を実現しているため，チェックポイントを取る必要がなく，原理的に任意の時点での移動が可能である．

4. 実　　装

3 章で述べた実現法に基づいて，Sun SPARC Station 上に実装を行った．

リモートメモリマップ・ファイル機構は，タスクスーパ・バイザをライブラリプログラムとして，ローカルキャッシュ・マネージャとファイルストア・マネージャは，UNIX デーモン・プロセスとして実装した．タスク・スーパバイザは，SunOS が提供している mprotect システムコールと singal システムコールを用いて SIGSEGV（Segmentation Fault）のシグナルハンドラとして実現した．マネージャ間およびマネージャとタスクの間の通信プロトコルは，TCP/IP を `TCP_NODELAY` オプションを設定して用いている．

テキストセグメントの再配置可能コードは，SunOS 共有ライブラリで用いられている Position Independent Code（PIC）の技術[2)]を用いて実装を行った．データセグメント内のアドレス空間依存情報に関する再配置情報は，モーバイルオブジェクトの最後にデータセグメントの各ページごとの再配置情報を記録する管理テーブル（ページ管理テーブル）を用意して，記録する．データセグメント内のアドレス空間依存情報の位置に関しても，コンパイラが出力するデバッグ情報をコンパイル時に静的に解析して，ページ管理テーブルに登録する．ページ転送時の再配置は，ページ管理テーブルからアドレス空間依存情報の位置を得る．

CPU-State セグメントは，スレッドの実行状態を表す，プログラムカウンタ，スタックポインタとスタックを格納する領域をコンパイル時に静的に作成する．実装に用いた SPARC アーキテクチャ（version 8）のスタックフレームは，**図 7** で示すような順序に基づいて積まれている[18)]．ローカル変数領域と引数領域は，シンボルテーブルの情報から，文献 6), 7) で述べられているスタック上の局所変数のデータ型の解析法を

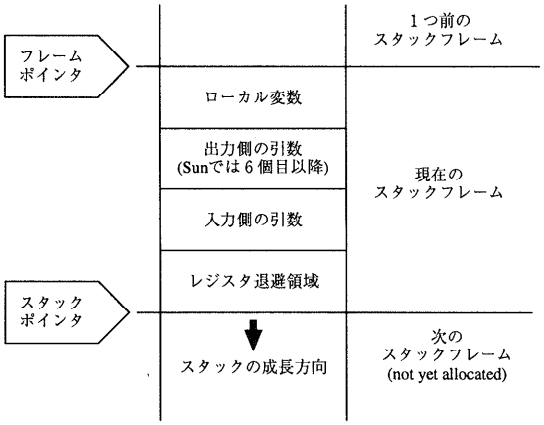
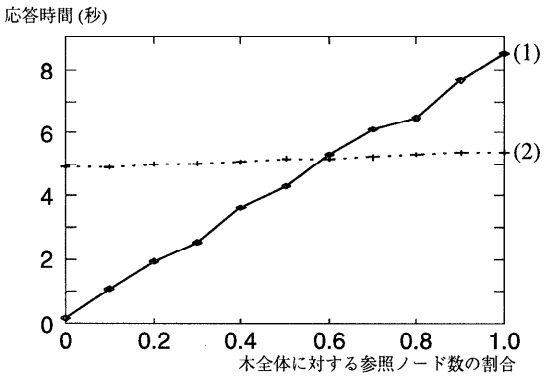


図7 SPARCアーキテクチャ（version 8）のスタックフレーム
Fig. 7 Stack frames of the SPARC architecture (version 8).



(1) リモートメモリマップ・ファイル機構
(2) TCP/IP ストリーム型ファイル転送機構

図8 実験1：リモートメモリマップ・ファイル機構の基本性能
Fig. 8 Experiment 1: Basic performance of the remote memory-mapping mechanism.

用いてアドレス空間依存情報の位置を調べる．レジスタが格納される領域は，レジスタ割当ての単位ごとにレジスタの割当て情報を出力するように既存のコンパイラに修正を加える．現在の実装システムは，レジスタとスタック内に動的に割り当てられるポインタ変数の反復的再配置を実装していない．

5. 実　　験

4章で述べた実装システムを用いて実験を行った．実験は，2台のSun SPARC Station（hyperSPARC 150 MHz, 64 MB RAM, SunOS 4.1.4）をEthernetで接続した環境を使用した．メモリマップ・ファイル機構で転送するページのサイズは，実験環境に使用したSunOSの仮想記憶ページングのサイズである4キロバイトを用いた．実験開始前に，毎回，DSRシステムのディスクキャッシュとオペレーティングシステムのメモリキャッシュをフラッシュした．実験値は，同一の実験を10回行った結果を平均している．

5.1 実験1：リモートメモリマップ・ファイル機構の基本性能

リモートメモリマップ・ファイル機構によるオブジェクト転送の効率化を検証するために，リモートメモリマップ・ファイル機構を用いてモーバイルオブジェクトの転送と参照を行った場合と，TCP/IPストリーム型ファイル転送機構を用いてそれらを行った場合の比較実験を行った．両方の場合について，サーバに1,048,575個（$2^{20}-1$個）のノードを持つ完全二分木が格納されているType Iモーバイルオブジェクトを作成し，クライアントがこの二分木を深さ優先で探索する時間を測定した．各ノードの大きさは4バイトである．実験結果を図8に示す．横軸は全ノード数に対する参照ノード数の比，縦軸は応答時間である．図8中の(1)は，リモートメモリマップ・ファイル機構によってファイル転送を行った場合の応答時間である．この場合，参照処理中に参照が発生したページのみが転送される．(2)は，TCP/IPストリーム型のファイル転送機構を用いて転送を行った後，参照を行った場合の結果である．この場合，参照処理に先立ち，オブジェクト全体が一度にストリーム転送される．

TCP/IPストリーム型ファイル転送機構(2)の場合は，参照するノードの個数にかかわらずオブジェクト全体が転送されるため，参照の割合が小さいときはリモートメモリマップ・ファイル機構(1)の方がTCP/IPストリーム型ファイル転送機構(2)よりも高速であることが期待される．逆に，参照の割合が大きくなるほど，転送オーバヘッドの少ないTCP/IPストリーム型ファイル転送機構(2)を使用した場合の方が効果的であると考えられる．本実験では，参照の割合が約60%のときを境界にして，それ以下のときはリモートメモリマップ・ファイル機構(1)が高速であり，それ以上のときはTCP/IPストリーム型ファイル転送機構(2)が高速であるという結果が得られた．

5.2 実験2：モーバイルオブジェクトのロード/アンロードにかかるオーバヘッド

DSRシステム上で，実行可能なプログラムコードを持つType IIモーバイルオブジェクトのロードおよびアンロードにかかるオーバヘッドを測定するために，プログラム（以下，$\mathcal{P}_{DSR}$と呼ぶ）を作成し実験を行った．サーバ計算機に格納されているType IIモーバイルオブジェクトをリモートメモリマップ・ファイル機構を用いてタスクの仮想記憶空間にロードしてスレッ

表1 実験2：プログラムの行数とコンパイル後のファイルサイズ
Table 1 Experiment 2: The number of lines of source files and the binary file size after compilation.

プログラム	$\mathcal{P}_{DSR}$		$\mathcal{P}_{UNIX}$	
	ソースコード（行）	バイナリコード（バイト）	ソースコード（行）	バイナリコード（バイト）
クライアントタスク	167	319,488	392	180,224
モーバイルオブジェクト	64	127,231	243	191,072

表2 実験2：モーバイルオブジェクトのロード/アンロードにかかるオーバヘッド
Table 2 Experiment 2: Overheads to load/unload a mobile object.

	計算内容：AB^{10}		計算内容：AB^{20}		計算内容：AB^{30}	
	$\mathcal{P}_{DSR}$ (ms)	$\mathcal{P}_{UNIX}$ (ms)	$\mathcal{P}_{DSR}$ (ms)	$\mathcal{P}_{UNIX}$ (ms)	$\mathcal{P}_{DSR}$ (ms)	$\mathcal{P}_{UNIX}$ (ms)
初期化	5.63	2.29	5.52	2.30	5.84	23.9
オブジェクトのロード	263.93	339.48	275.43	362.59	257.46	354.14
メソッドの検索	0.09	N/A	0.10	N/A	0.09	N/A
プログラムの実行	394.03	432.60	787.27	808.08	1173.04	1199.74
アドレス空間依存情報の再配置	3.51	2.78	3.45	2.83	3.41	2.77
オブジェクトのアンロード	199.64	152.10	198.80	148.81	196.61	150.67
応答時間	865.21	926.96	1269.06	1322.26	1635.02	1707.42

ドの実行を再開させた後，計算結果を含むモーバイルオブジェクトをサーバ計算機にアンロードする．実験に使用するモーバイルオブジェクトは，スレッドの実行が再開されるたびに行列 A，B に対し，

$$AB^n \quad (= A\underbrace{\cdot B \cdot B \cdots B}_{n})$$

を計算した結果を行列 A に代入する処理を行う．行列 A および B は，50 次の正方行列で，行列へのアクセスは同じ要素数のポインタ配列を用いて行う．行列とポインタ変数は，データセグメント内に確保してアンロードとともに永続化される．DSR システムを用いない場合の処理時間の指標を得るために，UNIX が提供する基本機能のみを使用して同様の内容のプログラム（以下，$\mathcal{P}_{UNIX}$ と呼ぶ）を作成し実験を行った．UNIX の場合のオブジェクト転送は，TCP/IP ストリーム型通信を使って実行に先だってオブジェクト全体をローカルディスクに転送するようにした．オブジェクトの実行は，fork システムコールで新しいプロセスを作成して exec システムコールを使って実行を行うこととした．プログラムの実行状態の永続化は，計算結果が格納されているデータセグメントをサーバサイトでデータファイルとして保存した．

実験のために作成した，$\mathcal{P}_{DSR}$ と $\mathcal{P}_{UNIX}$ のソースコードの行数およびコンパイル後のオブジェクトコードのサイズを**表 1** に示す．ソースコードは，C 言語を用いて作成したコードの行数である．オブジェクトコードのサイズには実行に必要なライブラリ群のサイズは含まれない．$\mathcal{P}_{UNIX}$ におけるモーバイルオブジェクト・プログラムのバイナリコードには，プログラムファイルのサイズ（131,072 バイト）と実行状態を保存するためのデータファイルのサイズ（60,000 バイト）を合計したサイズを記している．クライアントタスクのプログラムでは，$\mathcal{P}_{DSR}$ におけるコード行数が $\mathcal{P}_{UNIX}$ における行数の半分以下，モーバイルオブジェクトのプログラムにおいては約 4 分の 1 で記述可能であるという結果が得られた．これは，以下の 3 つの点において $\mathcal{P}_{DSR}$ の記述量が減ったことによる．

- リモートメモリマップ・ファイル機構によって明示的な入出力処理を記述する必要がない．
- モーバイルオブジェクトを動的にタスクにロードすることにより，プロシージャコールと同様の方法でオブジェクトに対するアクセスが記述可能である．
- DSR システムがヒープを含むデータセグメントとスタックとレジスタを含むスレッドの実行状態を永続化する機能を提供するため，実行状態の永続処理を明示的に記述する必要がない．

上記の $\mathcal{P}_{DSR}$ と $\mathcal{P}_{UNIX}$ を実行した実験結果を**表 2** に示す．ロードした際の計算を，AB^{10}，AB^{20}，AB^{30} の 3 通りの場合について結果を測定した．各項目の処理内容の詳細を以下に述べる．

初期化は，モーバイルオブジェクトをロードするための準備のための処理を行う．$\mathcal{P}_{DSR}$ では，モーバイルオブジェクトから利用可能なタスク内の変数や関数

に関するシンボル情報をメモリに読み込む処理を行う．$\mathcal{P}_{UNIX}$では，モーバイルオブジェクトが格納されている計算機のサーバに対してTCP/IPのセッションを確立する処理を行う．$\mathcal{P}_{DSR}$の主なオーバヘッドは，プログラムファイルからシンボルテーブルを読み込みハッシュテーブルに格納する処理である．

オブジェクトのロードは，モーバイルオブジェクトをサーバ計算機からタスクに転送する処理である．$\mathcal{P}_{DSR}$では，モーバイルオブジェクトのリモートメモリマップ，シンボル情報のロード，オブジェクトが必要とするライブラリのロードとページ管理テーブルのロードを行う．本実現のリモートメモリマップ・ファイル機構では，システム管理者によって設定された下限値パラメータに基づき，仮想記憶ページ単位とオブジェクト単位の2通りの転送方式を選択する．本実験では，オブジェクトのサイズが小さいため，実行に先だってオブジェクト全体がロードされている．オブジェクト$\mathcal{P}_{UNIX}$では，モーバイルオブジェクト・プログラムと行列データの転送およびローカルディスクへの格納を行う．

メソッドの検索は，モーバイルオブジェクト内のシンボル情報から実行するメソッド関数の先頭番地を検索する処理である．本実験では，オブジェクト内のメソッドの数が少ないために，非常に短い時間で検索が行えた．

プログラムの実行は，ロードしたモーバイルオブジェクトの実行を開始する処理を行う．$\mathcal{P}_{DSR}$の処理時間と$\mathcal{P}_{UNIX}$の処理時間の差は，$\mathcal{P}_{UNIX}$で行われている新しいプロセスの生成，転送された行列データファイルをメモリに読み込む処理，計算結果を同じファイルに書き戻す処理の3つのオーバヘッドが原因である．

アドレス空間依存情報の再配置は，ロードしたモーバイルオブジェクト内のアドレス空間依存情報の再配置処理を行う．$\mathcal{P}_{DSR}$では，ページ管理テーブルに登録されている，データセグメント，スタックおよびレジスタ内のアドレス空間依存情報の再配置を行う．また，その再配置情報をページ管理テーブルに登録する処理も行う．$\mathcal{P}_{UNIX}$では，データファイル中のポインタを新しくロードした仮想記憶空間に適応するように再配置を行う処理時間である．

オブジェクトのアンロードは，実行状態を含むモーバイルオブジェクトをサーバ計算機に格納する処理を行う．$\mathcal{P}_{DSR}$では，データセグメントやオブジェクト管理情報などのオブジェクト内の更新されたページとそのページの再配置情報を記録したページ管理テーブルを転送する処理を行う．$\mathcal{P}_{UNIX}$では，プログラムの実行結果である行列のデータを転送する処理を行う．

どの行列計算のオブジェクトに関しても，$\mathcal{P}_{UNIX}$よりも効率的にオブジェクトが実行できるという結果が得られた．どの場合も応答時間の差が小さいのは，オブジェクトのサイズが小さく，$\mathcal{P}_{DSR}$の場合も実行前にオブジェクト全体を転送しているためである．5.1節の実験1で言及したように，オブジェクトが十分に大きく，オブジェクト全体に対する参照の割合が半分以下になるようなアプリケーションの場合には，オブジェクトのロードに関するオーバヘッドを削減することができる．

6. 関連研究

本章では，モーバイルオブジェクトとリモートメモリマップ・ファイル機構のそれぞれに関連するシステムの特徴を述べ，DSRシステムとの違いについて述べる．

Java[3]は，同名のオブジェクト指向言語に基づいたプログラミング・システムである．JavaとDSRシステムの類似点および相違点は以下のとおりである．Javaでは，プログラムコードとデータを隠蔽したアプレットをサーバからアプリケーションへ動的にダウンロードする機能を提供する．この基本機能は，DSRシステムのType IIモーバイルオブジェクトを1度だけロードすることに相当する．DSRシステムでは，1つのオブジェクトに対してロードおよびアンロードを繰り返し行うことができ，また，オブジェクトもデータ（Type I），プログラムコード（Type II）に加えてプログラムの実行状態（Type III）を表現することができる．アプレットは，Javaコンパイラによって特定の機種に依存しないバイトコードで表現され，バイトコード・インタプリタによって実行される．DSRシステムでは，オブジェクトの実行時速度に焦点を置き，ネイティブコードによりオブジェクトの表現を行う．Javaでは，異なる計算機間でのオブジェクトの相互作用を行うための機構としてソケット通信機構を提供している．DSRシステムでは，オブジェクト自身が計算機間を移動することにより，異なる計算機間でのオブジェクトの相互作用を実現する．JavaRMI[21]，HORB[4]，Aglets[13]は，Javaを拡張することでDSRシステムにおけるType IIモーバイルオブジェクトのモビリティに相当する機能を提供している．

Telescript[22]は，同名のモーバイルエージェントの概念に基づいた分散プログラミングシステムである．データとプログラムを隠蔽したエージェントが，クライアントとサーバ間を移動・巡回する機能を提供する．

また，バイトコード・インタプリタが，オブジェクトの永続化機能を提供する．DSRシステムでは，タスク間でのモーバイルオブジェクトの移動をDSRを介して行うというモデルを提供することにより分散処理と永続処理を統合し，タスク間通信と永続オブジェクトを統一化している．

Emerald[5]は，同名の分散オブジェクト指向言語に基づいた分散システムである．データや実行モジュールを，すべてオブジェクトとして扱う．プロセスには，複数のオブジェクトが存在する．オブジェクトは複数サイト上のプロセスにまたがってモビリティを持つ．メモリマップ・ファイル機構が実現されていないが，プロセスより小さい単位（medium-grain）でモビリティを持たせている点は，DSRシステムと似ている．しかし，DSRシステムは，Emeraldシステムとは異なり，言語中立的である．また，Emeraldシステムでは，DSRシステムのようなオブジェクトの永続性は考慮されていない．

COOL[14]は，分散共有メモリとメッセージパッシングに基づいた分散システムである．オペレーティングシステムのレベルで分散共有メモリを実現することにより，システム全体でオブジェクトを共有することが可能となっている．オブジェクト間の通信はメッセージパッシングに基づいているが，分散共有メモリを使って通信先オブジェクトをメモリマップしてローカル通信に最適化することができる．オブジェクトのメモリマップをユーザ透明に行う点を除いて，メモリマップ・ファイル機構を使ってオブジェクトの永続性と分散性を提供する点はDSRシステムと類似している．DSRシステムでは，各計算機サイトの独立性を維持しつつ明示的にオブジェクトを移動させることにより，各サイトのアドミニストレーションの独立性や参加するサイトの動的な変化にも対応している．DSRシステムでは，オブジェクトの移動をすべてDSRを介して行うことで通信に永続性を与え，相手の状態に依存しない非同期的な通信が可能である．

ObjectStore[12]，QuickStore[23]，Texas[19]などの分散永続システムは，メモリマップ・ファイル機構を用いたC++オブジェクトのデータに関する分散処理と永続処理を提供する．プログラムコードやオブジェクトの計算状態に対する分散永続処理はサポートしていないか，オペレーティングシステムなど他のシステムの機能に頼っている．DSRシステムでは，データに加えて，プログラムコードや計算状態もfirst-class objectとして扱い，これらに対する分散永続処理を提供している．また，これらのシステムは，特定のプログラミング言語（C++）に強く依存した実現になっている．DSRシステムは，ミドルウェアレベルで実現を行っているためにプログラミング言語処理系には依存しない．そのため，既存の多くのプログラミング言語から利用可能である．

AL-1/D[17]は，メタレベル・プログラミングとリフレクションの技術を使ってオブジェクトの移動に関する方針を記述することができるプログラミング言語処理系である．バイトコードインタプリタに基づいたランタイムシステムを提供し，メタレベルプログラムによって記述された方針に基づいてオブジェクトのマイグレーションが行われる．これに対し，DSRシステムは，プログラムコードや計算状態を含めたオブジェクト移動のメカニズムを提供するミドルウェアに位置付けられるシステムである．

Apertos[25]は，オブジェクトの移動概念をオペレーティングシステムの実現に使用したシステムである．Apertosにおけるオペレーティングシステム・サービスは，オブジェクトマイグレーションを基本機構として用いることで実現されている．DSRシステムでは，オブジェクトの移動を分散処理と永続処理の統一的な実現に用いたシステムである．DSRシステムにおけるモーバイルオブジェクトは，一般的に，オブジェクト指向プログラミング言語におけるオブジェクトよりも大きな粒度（プログラムモジュールの粒度など）で用いる．

7. おわりに

分散永続性を提供するモーバイルオブジェクト・システムであるDSRシステムを，分散仮想記憶技術を用いることで効率的に実現する方法について述べた．この方法では，仮想記憶管理技術，ファイル管理技術，通信技術を組み合わせることにより，モーバイルオブジェクトのロードおよびアンロード操作時のデータ転送量を最少化している．モーバイルオブジェクトをロードする際に必要となるアドレスの再配置は，再配置可能コードとページ管理テーブルによって行う．この再配置技術は，再配置したアドレスの情報を永続化する際に付加することで，反復的な動的再配置を行うことができる．スレッドのアンロードは，スタック，レジスタ，プログラムカウンタを永続化することで実現できる．また，スレッドのロードは，スタック，レジスタとプログラムカウンタをデータセグメントの再配置と同様の方法で再配置することで新しくロードした位置に適合することができる．

今後の研究課題としては，第1に，異機種インタオ

ペラビリティの対応がある．現在，Sun SPARC アーキーテクチャと Intel Pentium アーキテクチャの 2 種類のプラットフォーム上で異機種インタオペラビリティの研究開発を行っている．データセグメントに関する異機種インタオペラビリティは，文献 11) によって述べられているヒープ領域の相互参照を含むデータを分散透明に受け渡す技術を参考として実現を行っている．オブジェクトの実行状態である CPU-State の異機種インタオペラビリティは，文献 20) を参考に検討を行っている．第 2 に，広域ネットワーク環境への対応がある．特に，ネイティブコードによるモーバイルオブジェクトの実行における保護とセキュリティ機構に注目して研究を進めている．現在，DSR システムの設計と実現を拡張し，広域ネットワーク環境のためのモーバイルオブジェクト・システム PLANET[8),9)] の開発を進めている．

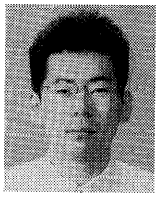
謝辞 有益な討論をしていただきました慶應義塾大学環境情報学部清木康先生，および筑波大学電子・情報工学系板野肯三先生に感謝いたします．また，本システムの実現にあたり，実装システムのデバッグや実験環境の整備に協力していただいた筑波大学工学研究科東村邦彦氏，染谷祐一氏に感謝いたします．

参考文献

1) Cattell, R.G.G.: *Object Data Management – Object-Oriented and Extended Relational Database Systems*, revised edition, Addison-Wesley (1994).
2) Gingell, R.A., Lee, M., Dang, X.T. and Weeks, M.S.: Shared Libraries in SunOS, White paper, Sun Microsystems (1993).
3) Gosling, J. and McGilton, H.: The Java Language Environment: A White Paper, Technical Report, Sun Microsystems (1995).
4) Hirano, S.: HORB: Distributed Execution of Java Programs, *Proc. Int. Conf. on Worldwide Computing and its Applications'97*, Tsukuba, Japan, p.A-1-2 (1997).
5) Jul, E., Levy, H., Hutchinson, N. and Black, A.: Fine-grained mobility in the Emerald system, *ACM Trans. Computer Systems*, Vol.6, No.1, pp.109–133 (1988).
6) Kato, K., Inohara, S., Narita, A., Chiba, S. and Masuda, T.: Design of the XERO open distributed operating system, *Journal of Information Processing*, Vol.14, No.4, pp.384–397 (1991).
7) Kato, K., Narita, A., Inohara, S. and Masuda, T.: Distributed shared repository: A unified approach to distribution and persistency, *Proc. 13th IEEE Int. Conf. on Distributed Computing Systems*, pp.20–29 (1993).
8) 加藤和彦，東村邦彦，松原克弥，相河 享，吉田 順，河野健二：モービルオブジェクトの概念に基づいた広域分散システム PLANET のシステムモデルについて，日本ソフトウェア科学会主催第 12 回オブジェクト指向計算ワークショップ WOOC96 論文集 (1996).
9) Kato, K., Toumura, K., Matsubara, K., Aikawa, S., Yoshida, J., Kono, K., Taura, K. and Sekiguchi, T.: Protected and Secure Mobile Object Computing in PLANET, *Proc. Int. Workshop on Mobile Object Systems*, Linz, Austria (1996).
10) 加藤和彦，益田隆司：分散 OS XERO：分散処理と永続処理の統一的な取扱いを目指して，情報処理，Vol.36, No.8, pp.708–714 (1995).
11) Kono, K., Kato, K. and Masuda, T.: Smart Remote Procedure Calls: Transparent Treatment of Remote Pointers, *Proc. 14th IEEE Int. Conf. on Distributed Computing Systems*, pp.142–151 (1994).
12) Lamb, C., Landis, G., Orenstein, J. and Weinreb, D.: The ObjectStore Database System, *Comm. ACM*, Vol.34, No.10, pp.50–63 (1991).
13) Lange, D.B., Oshima, M. and Kosaka, K.: Aglets: Programming Mobile Agents in Java, *Proc. Int. Conf. on Worldwide Computing and its Applications'97*, Tsukuba, Japan, p.B-4-3 (1997).
14) Lea, R., Jacquemot, C. and Pillevesse, E.: COOL: System Support for Distributed Programming, *Comm. ACM*, Vol.36, No.9, pp.37–46 (1993).
15) 松原克弥，加藤和彦：分散仮想記憶技術を用いた分散共有格納庫システムの実現法について，情報処理学会研究報告（SWoPP'94），Vol.94, No.64, pp.153–160 (1994).
16) 松原克弥，加藤和彦：分散永続性を提供するモービルオブジェクト・システムの実現法について，情報処理学会研究報告（SWoPP'96），Vol.96, No.79, pp.37–42 (1996).
17) Okamura, H. and Ishikawa, Y.: Object Location Control Using Meta-level-Programming, *Proc. 8th European Conference of Object-Oriented Programming*, LNCS, Vol.821 (1994).
18) Garner, R. and Weaver, D.R.T.: *The SPARC Architecture Manual Version 8*, Prentice-Hall (1992).
19) Singhal, V., Kakkad, S.V. and Wilson, P.R.: Texas: An Efficient, Portable Persistent Store, *Proc. 5th Int'l Workshop on Persistent Object System*, San Miniato, Italy (1992).

20) Steensgaard, B. and Jul, E.: Object and native code thread mobility among heterogeneous computers, *Proc. 15th ACM Symposium on Operating Systems Principles*, pp.68–78 (1995).
21) Sun Microsystems, I.: *Java Remote Method Invocation Specification* (1997).
22) White, J.E.: Mobile Agents, *Software Agents*, Bradshaw, J. (Ed.), MIT Press (1996).
23) White, S.J. and DeWitt, D.J.: QuickStore: A High Performance Mapped Object Store, *Proc. ACM SIGMOD Conf. on the Management of Data* (1994).
24) Wilson, P.: Pointer swizzling at page fault time: efficiently supporting huge address spaces on standard hardware, *ACM Computer Architecture News*, pp.6–13 (1991).
25) Yokote, Y.: The Apertos reflective operating system: The concept and its implementation, *Proc. ACM OOPSLA'92*, pp.414–434 (1992).
26) 吉田 順，東村邦彦，松原克弥，加藤和彦：分散ファイルシステムにおける名前空間の多重化について，日本ソフトウェア科学会第 13 回大会論文集，pp.237–240 (1996).

(平成 9 年 4 月 22 日受付)
(平成 10 年 6 月 5 日採録)



松原 克弥

1971 年生．1994 年筑波大学第三学群情報学類卒業．1996 年筑波大学大学院修士課程理工学研究科修了．同年同大学大学院博士課程工学研究科第 3 年次に編入学．1998 年筑波大学電子・情報工学系助手，現在に至る．分散システムの設計と実装に興味を持つ．

加藤 和彦（正会員）

1962 年生．1985 年筑波大学第三学群情報学類卒業．1988 年同大学大学院博士課程工学研究科中退．同年東京大学理学系研究科博士課程入学．1989 年同大学理学部情報科学科助手．1993 年筑波大学電子・情報工学系講師．1996 年同大学同学系助教授，現在に至る．1997 年より科学技術振興事業団研究員を併任．理学博士（東京大学）．オペレーティングシステム，プログラミング言語システム，データベースシステムなどシステムソフトウェア全般に興味を持つ．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE 各会員．