システムソフトウェアと
オペレーティング・システム 73－7
（1996．8．26）

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

松原 克弥† 加藤 和彦‡
† 筑波大学 工学研究科 ‡ 筑波大学 電子・情報工学系

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ビ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．モービルオブジェクトは，異なる仮想記憶空間の間を移動したり，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† Kazuhiko Kato‡
† Doctoral Program in Engineering, Univ. of Tsukuba
‡ Institute of Information Sciences and Electronics, Univ. of Tsukuba

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，目覚しい速度で広範な普及を遂げており，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．これに伴い，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，地理的に離れた計算機群をネットワークで接続し，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盛んに行われている．著者らは，このようなアプリケーションの構築をオペレーティングシステム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，以下，DSRシステム)の開発を行っている[2, 8, 5].

DSRシステムの最大の特徴の一つは，モービルオブジェクトの概念に基づいたプログラミングモデルを提供することにより，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で記述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プログラミングモデルは，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ビルオブジェクトと呼ぶ．モービルオブジェクトは，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ブジェクトおよび内部にスレッドをもつアクティブオブジェクトの両方を表現可能である．

以前に実装されたDSRシステムのプロトタイプシステム[2]では，モービルオブジェクトのロードおよびアンロード操作の際，モービ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，実質的にモービ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，モービ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ビ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一つであるメモリマップ・ファイル機能を分散環境に拡張した「リモートメモリマップ・ファイル機能」を実現し，モービ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小化する．さらに，実行可能モジュールをロードするためのアドレス空間依存情報(ポインタ)の反復的再配置(iterative relocation)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，CPUの実行状態を仮想記憶空間から取りだし，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，本論文は次のように構成されている．第2章では，第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．第3章では，分散環境に拡張したメモリマップ機構について述べ，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．第4章では，第5章で述べた方法で実装を行ったシステムを用いた実験について示す．最後に第5章で，まとめと今後の課題について述べる．

2 分散共有格納庫システムの概要

2.1 基本概念

DSRプログラミングモデルは，タスク，スレッド，モービルオブジェクト，DSRの4つの基本的な概念で説明される(図1参照)．

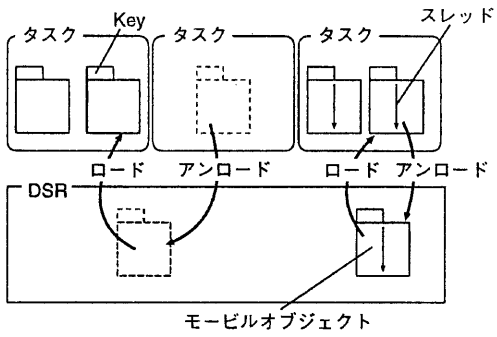

図1: 分散共有格納庫の基本概念

- 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空間からモービルオブジェクトをロード，アンロードして計算を行う．
- モービルオブジェクト：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．それぞれのモービルオブジェクトは，DSRシステム内では一意で位置独立なkey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．スレッドがモービ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．モービルオブジェクトは，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を介してモービルオブジェクトの受渡しを行う．DSR空間にあるモービルオブジェクトの実際の格納は物理的に分散されたサイトにすることも可能であるが，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．

1つのモービルオブジェクトは，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

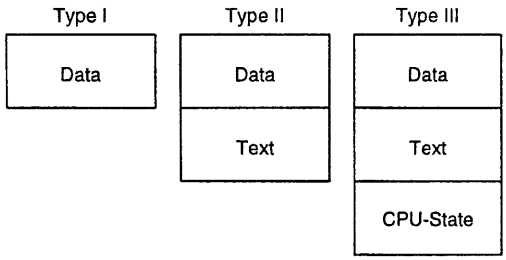

図 2: モービルオブジェクトの種類

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ビルオブジェクトが形成される．DSR システムで扱うモービ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 2参照)．Type I モービルオブジェクトは，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ビルオブジェクトとして扱われる．Type II モービルオブジェクトは，モービ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 モービルオブジェクトは，モービ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．このモービルオブジェクトは，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 モービ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．

3 分散仮想記憶技術を用いた実現

DSR システムにおけるモービ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，より効率的に行う実現法について述べる．本実現法には，以下の技術課題を解決する必要がある．

1. リモートメモリマップの分散透明な実現
2. アドレス空間依存情報 (ポインタ) の反復的な再配置技術の確立
3. 実行中のスレッドの動的なロード / アンロード技術の確立

1 のリモートメモリマップ機構の実現により，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．2 のアドレス空間依存情報の反復的再配置技術により，プログラムコードを含むコンテキスト (Type II, III) のロードおよびアンロードを実現することができる．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，内部にスレッドを含む Type III コンテキストのロードおよびアンロードを実現することができる．

1 のリモートメモリマップ機構の実現方法は，文献 [5] で報告を行った．本稿では，2 と 3 の技術課題に対する解決方法を述べる．

3.1 アドレス空間依存情報の反復的再配置の実現

DSR プログラミングモデルでは，1 つの仮想記憶空間 (タスク) の任意の位置にモービ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ビルオブジェクトを再び任意のタスクの任意の位置にロードできる．この機能を実現するために，ポインタなどのアドレス空間依存情報を反復的に再配置する機能 (反復的再配置) を実現する必要がある．以下に，テキストセグメントとデータセグメントにおける反復的再配置の実現方法を説明する．

3.1.1 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．再配置可能コードでは，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は，レジスタを用いた間接アドレッシングによって指定する．load 命令や store 命令で用いるアドレスは，データ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．

3.1.2 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，変数へのポインタなどのアドレス情報を，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，ポインタの位置情報の管理が必要になる．静的に割り当てられる領域は，コンパイル時に得られる変数の型情報を用いることで実現する．動的に割り当てられる領域は，Pascalなどの型制約の強い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，ポインタの内容を直接変更するために，実行時にすべてのポインタの再配置を行うと，本実装のリモートメモリマップ機構の特徴である必要最小限のデータ転送機能が生かされない．リモートメモリマップと反復的再配置を調和的に統合するには，参照が起きたページのみに対する再配置を行う必要がある．しかし，ページ毎に再配置を行うと，アンマップ時の書き戻しで，再配置が行われたページと行われていないページのアドレス参照に一貫性がなくなってしまう．従って，書き戻しの時に何らかの処理が必要となる．本実現では，データセグメント内のページのアドレス空間依存情報の位置とページに対する再配置処理の情報を保存するためのページ管理テーブルを用意することでこの問題を解決する．ページ管理テーブルは，各々のモービルオブジェクトに付加して，データセグメント内のページに含まれるアドレス空間依存情報の位置と，最も最近行った再配置処理の情報を記録する．記録されたページ管理テーブルは，対応するページがロードされる時にタスクに読み込まれる．

3.2 スレッドのロード/アンロード機構の実現

DSRシステムのモデルでは，モービルオブジェクトにバインドされたスレッドを，CPU-Stateセグメントによって表現する．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に相当する．CPU-Stateセグメントには，モービルオブジェクトにバインドされたスレッドの現在の実行状態や実行環境などの情報が入る．永続化されたType IIIモービルオブジェクトのロード時には，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．

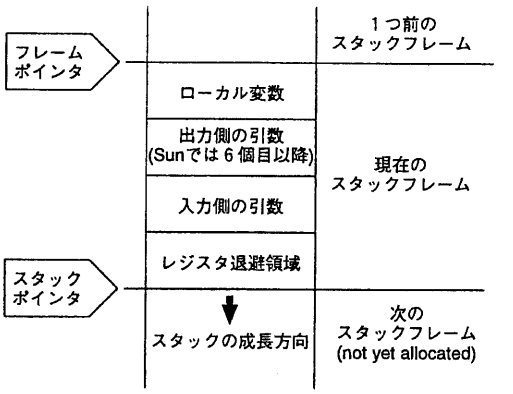

図 3: スタックフレーム

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，レジスタなどの情報が入っている．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．さらに，そのアドレス空間依存情報が格納されている場所や個数は，実行中に動的に変化する．DSRプログラミングモデルでは，モービルオブジェクトのアンロードはオブジェクト自身が明にプリミティブを発行する．プログラムカウンタとレジスタは，アンロードのプリミティブを発行した時にスタックに積まれるを利用することで，スタックの反復的再配置と同様の方法で再配置を行うことができる．スタック内のアドレス空間依存情報の位置は，スタックフレームの情報とシンボルテーブルの情報を用いて解析できる．実装に用いたSun SPARC Stationでのスタックフレームは，図3で示すような順序に基づいて積まれている．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[1, 2]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，実行状態によってアドレス空間依存情報の位置や数が変化するので，レジスタ割り当ての単位毎にレジスタの割り当て情報を出

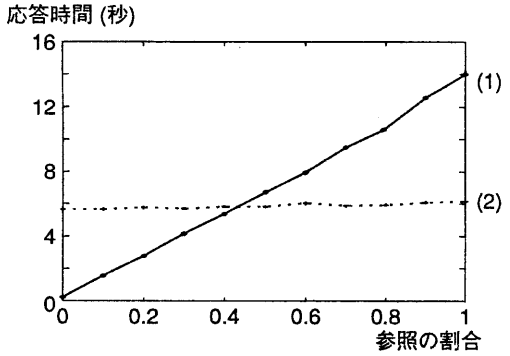

図 4: 完全二分木の深さ優先探索

力するように既存のコンパイラに修正を加える．調べたアドレス空間依存情報は，データセグメントの再配置と同様の方法で反復的再配置を行う．

4 実験

3章で述べた実現法に基づいて実装したシステムを用いて実験を行った．実験には，2台の Sun SPARC Station(microSPARCII 85MHz, 32MB RAM, SunOS 4.1.4) を Ethernet で接続した環境を使用した．メモリマップで転送するページのサイズは，SunOS の仮想記憶ページングのサイズである 4K 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ている．実験値は，同一の実験を 10 回行った結果を平均したものを用いた．

4.1 リモートメモリマップの基本性能

メモリマップ機構による実装の効率を検証するために，リモートメモリマップ機構を用いてファイル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を作り，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 4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 4中の (1) は，リモートメモリマップ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 によりストリーム型のファイル転送を行った後，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，ファイル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 (2) の場合は，参照するページの量に関わらずファイル全体が転送されるため，参照の割合が小さい時はリモートメモリマップ (1) の方が TCP/IP ストリーム型ファイル転送 (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程，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 (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 45% のときと境界にして，それ以上の時はリモートメモリマップ (1) が，それ以下の時は TPC/IP ストリーム型ファイル転送 (2) が高速であるという結果が得られた．にはそのような結果が得られた．

4.2 アクティブ・モービ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

オブジェクト内にスレッドがバインドされているアクティブ・モービ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した．

実験に用いた実装システムは，Pthread ライブラリ [6] を用いてユーザレベルスレッドを実現し，さらに 3.2節で述べた方法を用いてアクティブ・モービルオブジェクトのスレッド部分のロードおよびアンロード機構を実現している．

実験は，タスク上にアクティブ・モービルオブジェクトを生成して実行を開始した後，実行途中で DSR にアンロードする．さらに，アンロードしたアクティブ・モービルオブジェクトを別のタスクにロードして，アンロード前の状態を再現する．測定は，アクティブ・モービルオブジェクトのスレッド部分のロードおよびアンロード時間のみを測定するために，実際にロードおよびアンロードするセグメントを CPU-State セグメントのみとした．CPU-State セグメントには，スタックポインタ，プログラムカウンタ，シグナルマスク．エラー番号などとスタックが保存される．本実験でロードおよびアンロードを行った CPU-State セグメントの大きさは，約 16K バイトである．

表 1に実験結果を示す．実装に用いた Pthread ライブラリの性能を示すために，スレッドの生成，消滅，コンテキスト切替えの時間も測定した．実験結果では，アンロード処理のオーバヘッドが大きい．

表 1: アクティブ・モービルオブジェクトのスレッド部分のロードおよびアンロードと実装に用いたユーザレベルスレッドの性能

	処理時間 (msec)
スレッド部分のロード	2.03
スレッド部分のアンロード	7.08
スレッドの生成	1.23
スレッドの消滅	0.26
コンテキスト切替え	0.08

これは，ロード時の再配置処理を軽減するために，アンロード時に CPU-State セグメント内のアドレス空間依存情報を検索してページ管理テーブルに登録する処理を行っているためである．

5 おわりに

分散永続性を提供するモービルオブジェクト・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の設計および実現法について述べた．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，コンテキス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小化している．コンテキス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，異機種環境への対応がある．現在，異なる CPU の間でスレッドの移動を可能にする方法を，文献 [7] を参考に検討している．第 2 に，広域ネットワーク環境への対応がある．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分散モービルオブジェクト・システム PLANET [4, 3] の開発を進めている．

謝辞

有益な討論をして頂きました 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 肯三先生に感謝致します．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頂いた筑波大学 工学研究科 東村 邦彦氏に感謝致します．

参考文献

[1] K. Kato, S. Inohara, A. Narita, S. Chiba, and T. Masuda. Design of the XERO open distributed operating system. *Journal of Information Processing*, Vol. 14, No. 4, pp. 384–397, Apr. 1991. Special issue on new operating systems.

[2] K. Kato, A. Narita, S. Inohara, and T. Masuda. Distributed shared repository: A unified approach to distribution and persistency. In *Proc. IEEE 13th Int. Conf. on Distributed Computing Systems*, pp. 20–29, Pittsburgh, Pennsylvania, May 1993.

[3] K. Kato, K. Toumura, K. Matsubara, S. Aikawa, J. Yoshida, K. Kono, K. Taura, and T. Sekiguchi. Protected and secure mobile object computing in PLANET. In *Proc. of ECOOP Workshop on Mobile Object Systems*, Linz, Austria, 1996. To appear.

[4] 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 WOOC'96 論文集, 1996. 掲載予定.

[5] 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. SWoPP'94 情報処理学会研究報告 94-OS-65-20, pp. 153–160, Jul. 1994.

[6] Frank Muller. A library implementation of POSIX threads under UNIX. In *Proceeding of USENIX Winter Conference*. USENIX Association Conference Proceedings, Jan. 1993.

[7] Bjarne Steensgaard and Eric Jul. Object and native code thread mobility among heterogeneous computers. In *Proc. of the 15th ACM Symposium on Operating Systems Principles*, pp. 68–78, 1995.

[8] 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. 情報処理学会誌「情報処理」, Vol. 36, No. 8, pp. 708–714, 8 月 1995.