データベースシステム 113－17
（1997. 7. 14）

モーバイルオブジェクト・システム PLANET における分散永続オブジェクトの実現法について

松原 克弥† 加藤 和彦‡
† 筑波大学大学院 博士課程 工学研究科 ‡ 筑波大学 電子・情報工学系

近年，インターネットに代表される広域ネットワーク環境が急速に普及を遂げつつあり，広域ネットワーク環境を想定したアプリケーションの開発が盛んに行われている．広域ネットワーク環境は，大きな通信遅延の存在，不特定多数のユーザおよび計算機サイトの参加等，従来の分散プログラミングシステムでは想定されていなかった問題を数多くはらんでいる．我々は，ローカルエリアネットワーク環境と広域ネットワーク環境の両方の環境においてシームレスな分散アプリケーションの構築を支援するシステムとして，PLANET 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，PLANET システムの分散永続オブジェクトであるモーバイルオブジェクトを実現する方法について述べる．

Implementation of the Distributed Persistent Objects in PLANET system

Katsuya Matsubara† Kazuhiko Kato‡
† Doctoral Program in Engineering, Univ. of Tsukuba
‡ Institute of Information Sciences and Electronics, Univ. of Tsukuba

Nowadays worldwide networks such as Internet are becoming very poplar, so many academic and industrial research efforts have been made to develop applications for such networks. Compared to local area networks, worldwide networks have very different characteristics, such that the communication latency is large, and that the number of users and computer sites are very large and cannot be fixed in general. The authors are working on designing and implementing a distributed computing system for both local and worldwide networks. The system was named PLANET. One of the most notable feature of PLANET is that the system is designed based on the mobile object concept, by which we mean objects are separatable from a virtual address space. The mobile objects can go around different computer sites, and can be stored in persistent stores. This paper describes an implementation scheme of mobile objects in the PLANET system. Some experimental results are shown to validate the design of the scheme.

1 はじめに

近年，インターネットに代表される広域ネットワーク環境が急速に普及を遂げつつあり，広域ネットワーク環境を想定したアプリケーションの開発が盛んに行われている．広域ネットワーク環境は，大きな通信遅延の存在，不特定多数のユーザおよび計算機サイトの参加等，従来の分散プログラミングシステムでは想定されていなかった問題を数多くはらんでいる．我々は，ローカルエリアネットワーク環境と広域ネットワーク環境の両方の環境においてシームレスな分散アプリケーションの構築を支援するシステムとして，PLANET システムの開発を行っている [2, 4].

PLANET システムの最大の特徴の一つは，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実現できるようにしていることである．PLANET 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

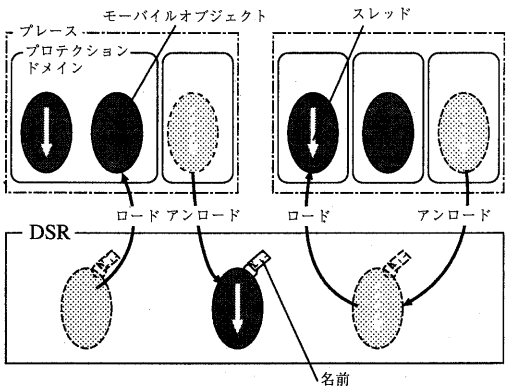

図 1: PLANET システムのシステムモデル

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．

本実現法では，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小化する．さらに，内部に実行可能コードを含むオブジェクトの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提案し，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実行を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，PLANET 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・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アプローチについて述べる．**第 4 章**では，実装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 PLANET システム

PLANET のシステムモデルは，**モーバイルオブジェクト**，**DSR** (Distributed Shared Repository; 分散共有格納庫の略)，**プレース**，そして**プロテクションドメイン**という 4 つの基本抽象概念を用いて説明される (図 1参照)．本章ではこれらの基本抽象概念を説明する．

計算処理の対象となるデータおよびデータに対する操作を密閉し，仮想記憶領域から分離可能にしたものを**モーバイルオブジェクト**と呼ぶ．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．PLANET 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．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，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・モーバイル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，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・モーバイル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．

PLANET のシステムモデルにおいて，モーバイルオブジェクトは **DSR** または**プレース**のいずれかに存在している．DSR は広域ネットワークと永続的記憶空間を抽象化したものであり，プレースはローカルエリアネットワークと揮発的記憶空間を抽象化したものである．DSR は広域ネットワークにより接続され，かつ，磁気ディスク装置に代表される永続的記憶装置を有したコンピュータ・サイト群により実現されることが想定されている．プレースはローカルエリアネットワークにより接続されたコンピュータ・サイト群により実現されることが想定されている．DSR を構成するサイトは永続的記憶装置を有することが必須であるが，プレースを構成するサイトはそうでない．

モーバイルオブジェクトを DSR からプレースに移動する操作をロード操作，逆にプレースから DSR に移動させる操作をアンロード操作と呼ぶ．全広域ネットワーク環境において DSR 空間はただ一つしか存在しないのに対し，プレースは数多く存在する．オブジェクトの実行は，オブジェクトがプレース上にあるときにのみ可能であり，DSR 上にあるときは可能でない．

プレース上の情報保護の単位を**プロテクションドメイン**と呼ぶ．一つのプロテクションドメインは必ず一つのプレースに属し，同時に二つ以上のプレー

スに属することはできない．オブジェクトがあるプレース上にロードされているとき，オブジェクトはそのプレース上の一つ以上のプロテクションドメインに必ず関連づけられている必要がある．個々のプロテクションドメインは独立した仮想アドレス空間を持ち，プロテクションドメインを乗り越えた情報アクセスを行うことができないようメモリ管理ハードウェアを用いて厳格に保護されている．

3 分散仮想記憶技術を用いた実現

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実現法について述べる．本実現法は，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・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プロテクションドメインとDSRの間で繰り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プロテクションドメインと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** 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および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** PLANET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．
- **データ転送量の最小化** 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小化する．
- **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，ページの更新が直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，プロテクションドメイン内のプロテクションドメイン・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(図2参照)．プロテクションドメイン・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のモーバイルオブジェクトに反映させる処理を行う．

3.2 アドレス空間依存情報の反復的再配置の実現

PLANETシステムのシステムモデルでは，1つの仮想記憶空間(プロテクションドメイン)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プロテクションドメイン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．

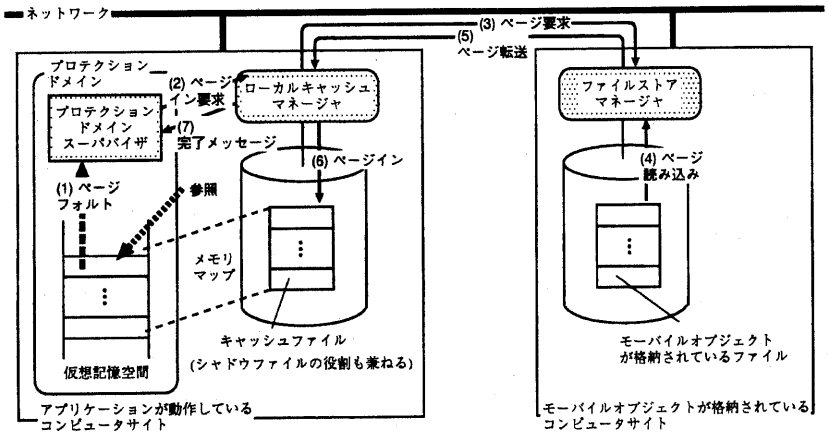

図 2: リモートメモリマップ・ファイル機構 (プロテクションドメインとモーバイルオブジェクトが別のコンピュータサイトにある場合)

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 [1] 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，仮想記憶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もに記録する方法を用いる．この方法は，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時のみページ単位で再配置処理を行う．再配置情報を記録するページ毎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が増加することによるオーバヘッドの増加が考えられるが，1 ページに必要な管理テーブルの大きさは 32 ビット CPU の場合で 4 バイトで，1 ページが 4 キロバイトとすると，仮想記憶ページの大きさで 1 ページ分の領域で 1024 ページ分の再配置情報が記録可能である．

3.3 スレッドのロード / アンロード機構の実現

2章で述べたように，PLANET 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．

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時は，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 [3]．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，データセグメントの場合と同様の再配置情報の管理テーブルを用いた反復的再配置処理を行う．

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．実験は，2台の Sun SPARC Station(hyperSPARC 150MHz, 64MB RAM, SunOS 4.1.4) を Ethernet で接続した環境を使用した．リモートメモリマップ・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，PLANET 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．

4.1 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

PLANET システム上で，オブジェクト内部にスレッドを持つ Type III 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，$\mathcal{P}_{\text{PLANET}}$ と呼ぶ) を作成し実験を行った．サーバサイトに格納されている Type III 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プロテクションドメインの仮想記憶空間にロードしてスレッドの実行を再開させた後，実行が進んだスレッドの状態を含むモーバイルオブジェクトを再びサーバサイトにアンロードする．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度に行列 A, B に対し，

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

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共に永続化される．

PLANET 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，$\mathcal{P}_{\text{UNIX}}$ と呼ぶ) を作成し実験を行った．PLANET システムが提供しているリモートメモリマップ・ファイル機構を用いた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．スレッドの再開は，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，$\mathcal{P}_{\text{PLANET}}$ と $\mathcal{P}_{\text{UNIX}}$ 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．$\mathcal{P}_{\text{UNIX}}$ 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トには，プログラムファイルのサイズ (32,768 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．クライアントとモーバイルオブジェクトの両方のプログラムにおいて，$\mathcal{P}_{\text{PLANET}}$ におけるコード行数が $\mathcal{P}_{\text{UNIX}}$ における行数の約半分以下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 $\mathcal{P}_{\text{PLANET}}$ 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プロテクションドメイン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．
- PLANET 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 $\mathcal{P}_{\text{PLANET}}$ と $\mathcal{P}_{\text{UNIX}}$ 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．

行列の計算オーバヘッドの大きさに関わらず，$\mathcal{P}_{\text{UNIX}}$ に対して $\mathcal{P}_{\text{PLANET}}$ の応答時間が長い．これは，プログラムの生産性を向上するために，$\mathcal{P}_{\text{UNIX}}$ で明示的に記述していたネットワーク転送処理と永続処理を，$\mathcal{P}_{\text{PLANET}}$ では仮想記憶管理機構を用いてシステムが行うためのオーバヘッドである．また，実験で使用したモーバイルオブジェクトが約 124 キロバイトと小さく，実行中にオブジェクト全体を参

表 1: プログラムの行数とコンパイル後のファイルサイズ

プログラム	$\mathcal{P}_{\text{PLANET}}$		$\mathcal{P}_{\text{UNIX}}$	
	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	139	13,436	392	15,128
モーバイルオブジェクト	101	127,231	188	92,768

表 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

	計算内容: AB^{10}		計算内容: AB^{20}		計算内容: AB^{30}	
	$\mathcal{P}_{\text{PLANET}}$ (ms)	$\mathcal{P}_{\text{UNIX}}$ (ms)	$\mathcal{P}_{\text{PLANET}}$ (ms)	$\mathcal{P}_{\text{UNIX}}$ (ms)	$\mathcal{P}_{\text{PLANET}}$ (ms)	$\mathcal{P}_{\text{UNIX}}$ (ms)
初期化	13.45	2.51	13.25	2.53	13.29	2.52
オブジェクトのロード	149.24	159.66	146.40	155.78	144.64	157.37
スレッドの再開	22.68	N/A	23.50	N/A	23.94	N/A
プログラムの実行	497.94	379.81	900.46	734.57	1295.10	1092.30
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	164.00	151.80	173.26	142.64	171.08	155.11
応答時間	851.08	694.24	1260.70	1035.97	1651.75	1407.76

照するため，オブジェクト全体に対する参照の割合が大きくなる場合にはTCP/IPストリーム通信型ファイル転送よりもリモートメモリマップ・ファイル機構の転送オーバーヘッドが大きくなってしまっているためである (文献 [5] のリモートメモリマップの基本性能の実験結果を参照).

5 おわりに

PLANET システムにおけるモーバイルオブジェクトのロード操作およびアンロード操作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小化している．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，異機種環境への対応がある．現在，異なるCPUを持つ計算機間でモーバイルオブジェクトの移動を可能にする方法を検討している．第2に，実行性能をさらに向上させるために，プリフェッチ技術を本稿で述べたリモートメモリマップ・ファイル機構に統合する研究を進めている．

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] Robert A. Gingell, Meng Lee, Xuong T. Dang, and Mary S. Weeks. Shared libraries in SunOS. White paper, Sun Microsystems, Inc, 1993.

[2] K. Kato. Safe and secure execution mechanisms for mobile objects. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems*, LNCS-1222, pp. 201–211. Springer-Verlag, 1997.

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

[4] 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 *Special Issues in Object-Oriented Programming*, pp. 319–326. Dpunkt-Verlag, 1997.

[5] 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), pp. 37–42, Aug. 1996.