Vol. 36 No. 8　　情　報　処　理　　Aug. 1995

解　説

日本におけるオペレーティングシステム研究の動向

1.3 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†

加藤和彦††　益田隆司†††

1. はじめに

ここ10年程度盛んに研究開発が行われてきた分散オペレーティングシステムは，ネットワークで結合された複数の計算機群に対する仮想化と資源管理により，分散アプリケーションプログラムの開発と実行を容易化することを目的として設計されている．有用な分散処理システムを構築しようとしたとき，地理的に分散した計算機間で情報の交換と共有を行うと共に，その情報に対して個々のアプリケーション・プロセスの生存時間とは独立な生存時間（しばしばプロセスの生存時間よりも長い生存時間を持つ）を持たせることが本質的に必要となることがしばしばある．そのようなアプリケーションの典型例は，電子メールシステム，ニュースシステム，あるいは，WWWに代表されるようなネットワーク情報検索システム等である．このように，プロセスの生存時間とは独立な生存時間を持つというデータの性質を**永続性**（persistency）と呼び，永続データに対する処理のことを**永続処理**と呼ぶ．近年，分散環境上においてユーザ間の協調処理を支援する分散協調アプリケーションの実現が重要な研究課題となっているが，これも分散処理と永続処理を同時に実現することが求められる典型的な応用分野である．

分散性と永続性は互いに直交した概念と考えられる（**図-1**参照）．分散オペレーティングシステムを含む従来のほとんどのオペレーティングシステムでは，分散性を司る機能はプロセス間通信機能として，また，永続性を司る機能はファイルシステムの機能として，それぞれオペレーティングシステム内の独立なサブシステムとして実現されてきた．しかし興味深いことに，ある共通点に着目すると，分散処理機能と永続処理機能に統一的な見方を与えることができる．その共通点とは，分散処理も永続処理も，プロセスの持つ仮想記憶空間の内部と外部の間の情報の移動が本質的となっているという点である．分散処理では異なるサイト上の異なるプロセス間での情報の移動が本質的であり，永続処理では揮発性を持つ記憶空間と永続性を持つ記憶空間の間での情報の移動が本質的である．分散オペレーティングシステムXEROは，この観点に着目し，分散処理機能と永続処理機能を統一的に実現することにより，分散協調処理環境のプラットフォームを構築することを目指して設計された．

一般にオペレーティングシステムはハードウェア環境を仮想化し，アプリケーションに対して一種の仮想機械を提供しているものと考えられる．そのような仮想機械の論理的な仕様をここではオペレーティングシステムのプログラミングモデルと呼ぶことにする．XEROのプログラミングモデルはハードウェア環境に加え，個々の実装に対しても独立性を保つことを配慮して設計された．この設計の概略を**2.**で述べる．**3.**では，このプログラミングモデルを実装する場合の技術課題の説明とそれに対する解決法を，筆者らがこれまでに行ってきた実装に基づき説明する．最後に**4.**で，現在進行中の研究課題について述べる．

2. XEROのプログラミングモデル

XEROのプログラミングモデルでは，仮想アドレス空間上の任意のデータを，その仮想アドレス空間から切り離して永続記憶空間に格納した

† Unified Treatment of Distribution and Persistency in the XERO Operating System by Kazuhiko KATO (Institute of Information Sciences and Electronics, University of Tsukuba) and Takashi MASUDA (Department of Information Science, Graduate School of Science, University of Tokyo).

†† 筑波大学電子・情報工学系
††† 東京大学大学院理学系研究科

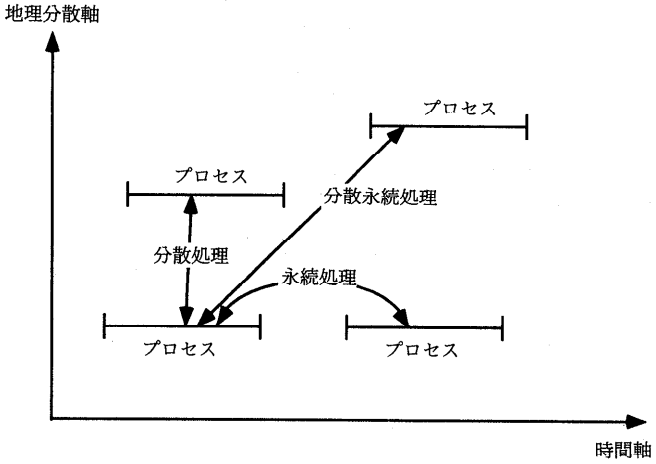


図-1　分散処理と永続処理

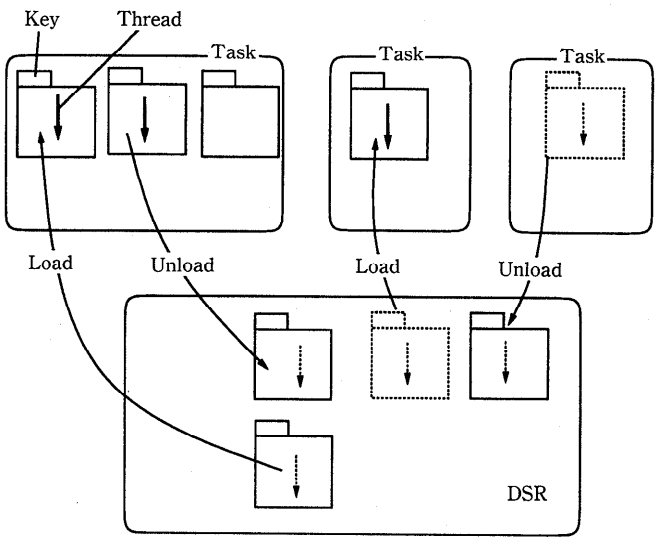


図-2　XERO のプログラミングモデル

り，異なるサイト上の異なるアドレス空間に転送することができる．このデータはバイナリプログラムやアドレス参照情報を含むデータであってもよく，さらにスレッドの実行状態までもこのデータに含めることができる．これにより，XERO では仮想記憶空間と永続記憶空間の間を行き来するデータ，および，分散環境上を行き来するデータを統一的に操作するプログラミング環境をユーザに提供することが可能になっている．それに加え，スレッドの実行途中でその状態を永続記憶空間に格納し，それを別のサイトの仮想アドレス空間に転送し，そこでスレッド実行の継続を行わせることまでもが可能になっている．

2.1 基本概念

XERO のプログラミングモデルは，**タスク**，**コンテキスト**，**スレッド**，そして**分散共有格納庫**（Distributed Shared Repository；以下，DSR と略す）という 4 つの概念を用いて述べることができる．

● **タスク**は，情報に対する処理を行うための保護された作業場であり，1 つの線形的にアドレッシングが可能な仮想アドレス空間である．

● **スレッド**は，仮想化された CPU であり，1 つのタスク上に，同時に複数のスレッドが存在可能である．

● **コンテキスト**は，システムによる情報管理の

単位である．プログラムやデータに加え，計算の実行状態（従来のシステムにおけるプロセスのスナップショット・イメージに対応）もコンテキストとして管理される．

● **DSR** は，コンテキストを永続的に格納するための格納庫である．コンテキストをDSRに格納する際，分散処理環境上で単一で物理位置独立な論理名と，アクセス保護情報をそのコンテキストに付加する．

コンテキストをDSRからタスクへ移動させることをコンテキストの**ロード**操作，逆にコンテキストをタスクからDSRへ移動させることを**アンロード**操作と呼ぶ．アンロードされたコンテキストは，そのコンテキストの論理名を知っていて，かつ，アクセス保護情報に関して違反を起こさない分散処理環境上のタスクにロード可能である．

1つのコンテキス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．データセグメントとは書込み可能なメモリ領域であり，テキストセグメントから操作が行われる．テキストセグメントはCPUによって直接実行されるメモリ領域である．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，レジスタ値などが保持される．コンテキストにおけるセグメントの組合せは2^3通り考えられるが，通常はそれらの組合せのうち，次の3つが重要である．それら3つのタイプのコンテキストを，セグメント数に応じ，**タイプI**，**II**，**III**のコンテキストと呼ぶ．これら3つのタイプは，

タイプI⊂タイプII⊂タイプIII

という包含関係にある．**タイプI**コンテキストは内部にデータセグメントのみを持ち，CPUにより直接実行される実行コードは含まれない．文書データ，グラフィックイメージデータなどはタイプIコンテキストとして扱われる．**タイプII**コンテキストは内部にデータセグメントとテキストセグメントを持つ．プログラミング言語のコンパイラにより出力されるライブラリ，抽象データ型(abstract data type)，あるいはパッシブ・オブジェクトを表現するのに使われる．**タイプIII**コンテキストはデータセグメント，テキストセグメントに加え，CPU-Stateセグメントを内部に持つ．リンカによって生成されるロードモジュールや，プログラムの実行イメージ，あるいはアクティブ・オブジェクトの表現にタイプIIIコンテキストを使用できる．

各コンテキストがDSR上で持つ名前の論理空間が単一であることを利用すると，この名前空間を用いることにより，分散して存在するタスク上のスレッド間の同期をとることが可能である．XEROのプログラミングモデルでは，論理的な名前を物理的な名前に変換する際に同時に分散同期制御を行うという，新たに考案された名前解決機構に基づいてシステム・プリミティブの設計が行われた．この機構を直感的に説明すると，各タスクからDSRの名前空間中のある名前を見ることができるか否かによって同期をとるというものである．この機構については**3.3**でより詳しく説明する．

3. 実現技術

3.1 “offshore”カーネル

XEROのプログラミングモデルにおいては，タスクは1つの仮想的な計算機と考えられる．なぜなら，1つの空間に互いに独立な複数のコンテキストを読み込み，それらのコンテキスト上に独立にスレッドを走行させることができるからである．我々が行ったXEROの実装では，タスクの中にそのタスク内の計算の実行を管理する**タスクスーパバイザ**を設け，タスクスーパバイザとカーネルとの協調によってシステム全体の管理を行うこととした（**図-3**参照）．タスクスーパバイザは，ユーザアドレス空間中に存在し，CPUのユーザモードで実行されるプログラムモジュールである．タスクスーパバイザは，従来のオペレーティングシステムのカーネルの機能の一部を，ユーザアドレス空間に移動させたものと考えられ，我々はこのアプローチを“offshore”カーネルアプローチと呼んでいる．

“offshore”カーネルアプローチはオペレーティングシステムの設計上，次の3点で有用である．第一に，カーネルの機能の一部をユーザアドレス空間に移動できるため，カーネルの大きさを抑制することができる．第二に，カーネルの力を必要とせず，タスクスーパバイザだけで処理を行える場合は，ユーザモードとカーネルモードとの間の文脈切り換えを起こすことなく処理できるので，

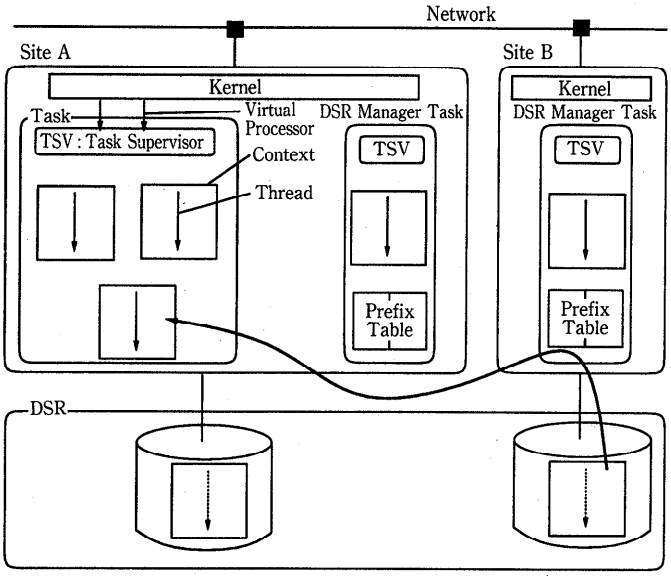


図-3 XEROの実現

システムオーバヘッドの軽減に貢献する．第三に，ユーザやアプリケーションに特化したタスクスーパバイザを設計することが可能となり，ユーザやアプリケーションの知識を活かした，柔軟で効率的なシステム機能の実現が可能となる．

3.2 マルチスレッドの実現技術

“offshore”カーネル・アプローチを採用したことにより，マルチスレッドの効率的な実現を興味深い手法を用いて行うことが可能になった．従来のオペレーティングシステムにおけるマルチスレッドの実現法は2つの方法に大別される．1つは，マルチスレッドをユーザアドレス空間内だけでコルーチンのように実現するもので，たとえばSunOSのLight-WeightProcess機能がこれに該当する．この場合，カーネルによって割り当てられた仮想プロセッサをユーザアドレス空間内のスケジューリングによって多重化し，複数のユーザスレッドを実現する．もう1つの方法は，1つのユーザアドレス空間に対して，カーネルが複数の仮想プロセッサを割り当てられるようにするもので，Machのマルチスレッド機能がこれに該当する．

これら2つの方法は対照的な長所・短所を持っている．前者では，ユーザアドレス空間内のみでマルチスレッドを実現しているために，スレッド間の文脈切換えが高速であるが，1つのスレッドがシステムコールを発して実行がブロックされると，同一ユーザ空間内の他のすべてのスレッドまでブロックしてしまう．また，プリエンプティブなスレッドスケジューリングを一般的には実現できない．一方後者では，カーネルが提供する仮想プロセッサとスレッドとが直接的に結びついているために，スレッド間の文脈切り換えの際にカーネルの介在が必要であり，スレッド間の文脈切り換えオーバヘッドは比較的に大きい．しかし，ユーザ空間内のあるスレッドがシステムコールを発行して実行がブロックされても，同一ユーザ空間内のスレッドの実行には影響を与えない．また，プリエンプティブなスレッドスケージューリングが実現可能である．

“offshore”カーネル・アプローチを用いると，これら2つの長所を組み合わせたマルチスレッド機構の実現が可能となる．XEROでは，1)スレッドの生成，消滅，文脈切り換えを高速に実現しつつ，2)あるタスク内のスレッドがシステムコールを発行して実行がブロックされても，そのタスク内の動作可能なスレッドの実行がブロックすることはなく，そして，3)プリエンプティブなスケジューリングが行われるマルチスレッドを提供している．この技術の詳細については文献1)，2)を参照されたい．

3.3 分散共有格納庫（DSR）の実現技術

XEROのプログラミングモデルは，分散ファイルシステムに同期機構を取り入れ，さらにそれ

を仮想記憶管理と統合している点に特徴がある．XEROのプログラミングモデルを実現するために必要で，これまでのオペレーティングシステム技術にない主要技術は，(a)同期をともなう分散名前解決機構と(b)コンテキストの反復的再配置機構である．以下これらの機構について解説する．

分散協調処理アプリケーションを構築する上で必要とされる機能の中で最も基本的でかつ重要な機能は，地理的に離れたタスク間での効率的な共有資源アクセス法と，それら分散したタスク間での効率的な分散同期制御法である．効率的な教習資源アクセスを行う上で重要な点は，資源に与えられた論理的な名前からその資源の物理的位置を効率的に求めるという分散名前解決機構の実現に関するものである．XEROでは，共有資源はすべてDSRに格納されているので，XEROにおける分散名前解決機構はDSRのそれに集中している．XEROでは，一般的には別々の機構として用意されることが多い分散名前解決機構と分散同期制御機構を統一的な枠組みのもとで実現するというアプローチをとった．このアプローチのアイデアは，DSR上の資源（コンテキスト）に与えられた名前そのものをセマフォのように考え，名前解決の成功をもって同期の成立に代えるということである．すなわち，名前を獲得すると同時にその名前をDSRの名前空間から隠蔽することにより，その間，その名前に対する他のユーザタスクによる名前解決要求（同期要求）はブロックされる．

分散協調アプリケーションでは，同期をともなないながらの共有資源アクセスが頻繁に行われるが，この同期をともなう分散名前解決機構によりアプリケーションプログラムの記述が容易化される．またこの機構は，効率的な分散名前解決法として提案された**プレフィックス・テーブル**(prefix table) 法[9]と親和性がよく，プレフィックス・テーブル法を拡張した方法を用いることにより，一般には容易ではない分散同期機構の効率的な実現を比較的に簡単に実現できる．この実現法では，同一の名前解決機構を各クライアントが持ち，コンテキストを永続的に格納するサーバ群と通信し合いながら，分散協調的に名前解決を行う．名前解決によって得られた結果は各クライアントが保持するプレフィックス・テーブル中にキャッシュされ，2回目以降の名前解決においてこのテーブル内のキャッシュがヒットする場合は，高速に名前解決処理が行われる．この名前解決の過程で，指定した名前に対応するコンテキストの実体を格納すべきサーバサイトは一意に定められ，分散同期制御の問題は，この一意に定められたサーバ上での同期制御問題に簡約化される．

(b)の反復的再配置機構は，コンテキストをDSRとタスクの間で繰り返し移動させるために必要な機構である．コンテキストを反復的に再配置するためには，コンテキストが内部に含むテキスト（プログラムコード）セグメント，データセグメント，CPU-stateセグメントをそれぞれ反復的に再配置可能とすればいい．

テキストセグメントの再配置は，従来の反復のない，一回限りの再配置機構で用いられる手法の1つである，レジスタ相対アドレッシングモードを用いたプログラムコードを生成するコンパイラを用いることで達成できる．

データセグメントの再配置をするために必要なことは，データセグメント中にあるアドレス空間依存情報（すなわちポインタ）の位置を把握し，ロードされたタスク中での位置に応じて，それらを正しく補整することである．データセグメントは静的データ領域とヒープ領域（動的データ領域）とで構成されている．静的データ領域はコンパイラによって生成されるシンボルテーブルに含まれる情報をもとにして，ポインタの位置を把握できる．ヒープ領域に関しては，Pascal言語などの静的に強く型付けされた言語の場合は，既存のコンパイラに若干の修整を施すことで，ヒープ領域のどこにポインタがあるかをコンテキスト生成時にコンテキストに付加させることができる．C言語などの型付けを動的に行う機能（キャスト機能やユニオン型など）を持つプログラミング言語を用いる場合はプログラム（プログラマ）に実行時にヒープ領域の型付け情報を提供してもらい，その型付け情報からポインタの位置を把握する．この型付け情報の提供は，タスクスーパバイザが提供するシステムプリミティブを呼び出すことにより行っている．

CPU-stateセグメントはCPUの実行履歴を格納するスタック領域とCPUのレジスタ，および

プログラムカウンタを内部に持つ．スタック領域は，コンパイラによって生成されるシンボルテーブル情報とスタックフレームの動的な解析[3]により反復的再配置が可能である．プログラムカウンタの補整は，データセグメント中のポインタの再配置と同じ要領である．レジスタの再配置は，コンパイラが最適化のために複雑な使い方をすることが多いため，一般性のある実現は容易でない．現在の実装では，コンテキストがタスクからDSRにアンロードされる可能性がある個所*ではレジスタ上にアドレス情報を残さないようにコンパイラのコード生成法を修正することでこれに対処するようにしている．

4. おわりに

XEROの実装はSONY NEWSワークステーション上に行われた．研究努力を集中させるためにUNIX 4.3 BSDカーネルのソースコードに修整および付加を行っていくという形で実装は進められた．その初期の設計は文献3）で述べられている．そこで述べられた二次記憶管理を効率的に行うための新たな技術として**永続キャッシング**技術が開発された[4]．XEROのマルチスレッド管理技術の詳細は文献1），2）で述べられている．文献3）で述べられた初期の設計はその後改良が加えられ，DSRシステムとして文献5）にて提案された．

現在，以下のような研究開発を行っている．第一に，仮想記憶管理をネットワークワイドに拡張した分散仮想記憶技術を用い，タスクへのコンテキストのロードおよびアンロードを効率化する方法を研究している[7]．第二に，DSRの名前空間を多重化させ，DSRを柔軟に運用できるようにする方法を研究している[6]．第三に，2.で述べたプログラミングモデルの上位層プログラミングシステムとして，モービル（可搬性）オブジェクトの概念に基づいた分散オブジェクトシステムの研究を行っている[8]．

謝辞 本稿は，東京大学大学院理学系研究科の猪原茂和氏，成田篤信氏（現在NTT（株）），坂田尚也氏（現在NTTデータ通信（株））らとの共同研究に基づいている．

NEWSワークステーションの技術情報を提供していただきましたソニー（株）に感謝いたします．

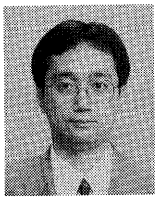
本稿の原稿に対して，有益なコメントを下さいました閲読者の方に感謝いたします．

* Type IIIコンテキストのアンロードは任意の時点で可能なのではなく，アンロードされるコンテキスト上のスレッドが，タスクスーパバイザが提供するアンロードのためのシステムプリミティブを明示的に発行したときのみアンロード可能である．このプリミティブの呼出し個所を検出してアンロードされる可能性のある個所を見つけている．

参考文献

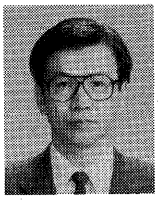
1) Inohara, S., Kato, K. and Masuda, T.: Unstable Threads' Kernel Interface for Minimizing the Overhead of Thread Switching, In Proc. of the 7th IEEE Int. Parallel Processing Symp., pp. 149 155 (Apr. 1993).
2) Inohara, S., Kato, K., Narita, A. and Masuda, T.: A Thread Facility Based on User/Kernel Cooperation in the XERO Operating System, In Proc. IEEE 15th Int. Computer Software and Applications Conference, pp. 398-405, Tokyo (Scp. 1991).
3) 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 (Apr. 1991).
4) Kato, K. and Masuda, T.: Persistent Caching: An Implementation Technique for Complex Objects with Object Identity, IEEE Trans. Software Engineering, Vol. 18, No. 7, pp. 631-645 (July 1992).
5) Kato, K., Narita, A., Inohara, S. and Masuda, T.: Distributed Shared Repository: A Unified Approach to Distribution and Persistency, In Proc. IEEE 13th Int. Conf. on Distributed Computing Systems, pp. 20-29 (May 1993).
6) 加藤和彦，坂田尚也，益田隆司：分散共有格納庫への多重名前空間の導入について，コンピュータシステムシンポジウム予稿集，情報処理学会，pp. 115-122 (Oct. 1993).
7) 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，Vol. 94, No. 64, pp. 153-160 (July 1994) (SWoPP'94).
8) 東村邦彦，加藤和彦，松原克弥：オブジェクト・モビリティに基づいた分散プログラミングシステムについて，尾内理紀夫(編)，オブジェクト指向コンピューティングIV—日本ソフトウェア科学会WOOC' 95，近代科学社，1995（刊行予定）．
9) Welch, B. and Ousterhout, J.: Prefix Tables: A Simple Mechanism for Locating Files in a Distributed System, In Proc. IEEE Int. Conf. on Distributed Computing Systems, pp. 184-189 (1986).

（平成6年11月7日受付）



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，現在に至る．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，ACM，IEEE各会員．



益田 隆司（正会員）

1939年生．1963年東京大学工学部応用物理学科卒業．1965年同大学院修士課程修了．同年（株）日立製作所入社．1977年から筑波大学，1988年から東京大学に勤務．現在，同大学院理学系研究科情報科学専攻教授．専門はオペレーティングシステム．