2G−7

分散OS XEROにおける複合オブジェクトに基づいたファイルシステムについて

加藤 厚志*　猪原 茂和**　加藤 和彦**　益田 隆司**
*(株) 日立旭エレクトロニクス　** 東京大学 理学部 情報科学科

1 はじめに

現在普及しつつある分散処理環境は、単一なハードウエア・アーキテクチャの計算機群により構成されていることはまれであり、むしろさまざまなアーキテクチャを持った計算機により構成されていることが多い。このような異機種分散処理環境におけるオペレーティングシステムでは、次の4つの分散性に対応することが必要であると我々は考えている[加藤91]。

- **地理的分散性:** 異なる地点に置かれたサイト上のプログラム間で情報の交換と共有を行う機能。これまでの分散オペレーティングシステム研究において言われていた分散透明性がこれにあたる。
- **ハードウエア・アーキテクチャ的分散性:** 異なるハードウエア・アーキテクチャを持つ計算機間で情報の交換と共有を行う機能。
- **ソフトウエア・アーキテクチャ的分散性:** 異なる開発環境で開発されたプログラム間で情報の交換と共有を行う機能。
- **時間的分散性:** 異なった時点のプログラム間で情報の交換と共有を行う機能。

このような分散性に対応することを目標として、我々は分散オペレーティングシステムXEROの開発を進めている。本稿では、XEROの永続オブジェクト管理系(Persistent Object Manager; POM)の設計と実現について述べる。POMは、地理的分散、ファイルシステムの基本機能である時間的分散に加え、異なるハードウエア・アーキテクチャの計算機間、異なる開発環境で開発されたプログラム間での情報の共有を可能にする。そのためPOMでは、2次記憶上のデータを、単なるバイト列としてではなく複合オブジェクトの概念に基づいて管理する。複合オブジェクトは、データと、そのデータの型情報を組にして管理したものである。各オブジェクトはオブジェクト識別子(Object Identifier; OID)によりアクセスされ、あるオブジェクト内に他のオブジェクトのOIDを保持することにより、オブジェクト間の複雑な参照関係を管理することが出来る。

2 永続オブジェクト管理システムの概要

POMは、2次記憶上のデータを複合オブジェクトの概念に基づいて管理する。複合オブジェクトは、OID、型情報、値の3要素によって構成される。OIDは、オブジェクトを一意に識別するための値である。その値はシステム内でユニークであり、1つのOIDはただ1つのオブジェクトを特定する。OIDの値は、オブジェクトのいかなる論理的、物理的な変化によっても変化しない。

従来の多くのシステムでは2次記憶上のオブジェクトの識別をユーザが与える名前で行っているが、これに対し本システムでオブジェクトの識別に用いるOIDは固定長である。そのため、2次記憶上でのOIDの格納や操作を容易に行うことが出来る。これにより、オブジェクト間の複雑な参照関係の効率的な実現が可能となる。

オブジェクトの型情報として、現在本システムでは以下の5つの型をサポートする。

- **atom型:** 整数型、実数型といった単一要素の型。float、double、short、int、long等をサポートする。
- **byte_sequence型:** POMでその内部を型付けしないバイト列。画面情報、音声情報など構造を持たないデータや、UNIXでいうファイルの概念を表現するためにこの型を用いる。
- **tuple型:** 異なる型を持つオブジェクトのいくつかをひとまとめにしたもの。C言語のstruct、Pascalのレコードに対応する。
- **set型:** 同じ型を持つオブジェクトの集合。その要素数は可変であり、オブジェクト生成後に要素の追加および削除を行うことが出来る。
- **array型:** 同じ型を持つオブジェクトの集合。その要素数は固定であり、オブジェクト生成後に要素の追加および削除を行うことは出来ない。要素には順序がついており、要素へのアクセスはその要素が先頭から何番目かを指定して行う。
- **union型:** 複数の型のオブジェクトのいずれかを要素として持つ型。

tuple型、set型、array型、union型は型構成子であり、これらを自由に組み合わせてユーザの必要とするデータ構造を表現する。また、byte_sequence型も型構成子であり、byte_sequence型オブジェクトのサイズはユーザが自由に定義する。生成された後のbyte_sequence型オブジェクトはatom型オブジェクトとして取り扱われる。これらの型情報は、型情報識別子(Type Identifier; TID)により識別される。TIDはOID同様にシステム内でユニークであり、1つのTIDはただ1つの型情報を特定する。

3 永続オブジェクト管理システムの内部構造

3.1 オブジェクトの内部表現

各々の型のオブジェクトは、POM内部ではそれぞれ以下にのように表現されている。atom型オブジェクト、byte_sequence型オブジェクトはともに要素が単一で、値は固有である。そのため、オブジェクトに値を直接格納する。tuple型オブジェクト、およびset型オブジェクトは、オブジェクト内にその要素となるオブジェクトのOIDを格納することにより表す(図1)。型の要素にatom型データ、byte_sequence型データが存在する場合、その要素はOIDのかわりに値を直接格納する。union型オブジェクトには、共用させる型のなかで最大サイズのエリアを確保しておく。union型が格納するオブジェクトの型もPOMが管理する。データをアクセスする際は、格納されているオブジェクトの型をもとにアクセスサイズを決定し、必要な部分にアクセスを行う。

tuple型、set型、union型オブジェクトの別の表現方法として、以下に述べる2つが考えられる。まず1つ目として、これらのオブジェクトの要素の如何にかかわらずオブジェクト内に直接値を格納する方法である。この表現方法はデータの参照効率の面では優れているが、1つのオブジェクトを複数のtupleの要素とするなどのオ

Persistent Object Management System in the XERO Distributed Operating System, by Atsushi KATO*, Shigekazu INOHARA**, Kazuhiko KATO**, and Takashi MASUDA**
Hitachi Asahi Electronics Co., Ltd.
*Dept. ofInformation Science, Univ. of Tokyo

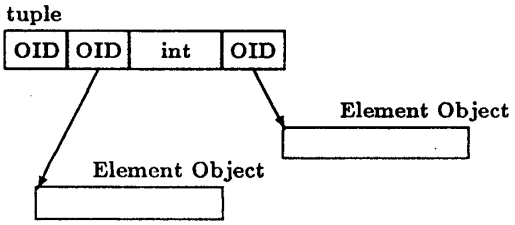

図 1: tuple 型オブジェクトの例

ブジェクトの共有が行えず、複雑なデータ構造を表現することが出来ない。そのためにこの方法は用いず、上に述べたような表現方法をとっている。2つ目として、永続キャッシング技術 [Kato92] を用いる方法である。この方法は以上で述べたオブジェクトの管理方法を発展させたものであり、参照する側のオブジェクトが存在するディスクページ内に、参照される側のオブジェクトの複製やその物理アドレスを保持する。これにより、2次記憶上のオブジェクト間の参照関係をたぐるコストを軽減させることが出来る。今後この技術を POM に取り入れる予定である。

3.2 永続オブジェクト管理システムの内部処理

POM 内部のシステム階層は、ディスクブロック管理層、可変長エリア管理層、複合オブジェクト管理層の3層からなる。ディスクブロック管理層は、POM の最下層であり、ディスクブロックの確保、解放、参照、更新などの2次記憶装置の管理を行う。可変長エリア管理層は、POM の中間層であり、ディスクブロック内の可変長データの確保、解放、参照、更新などの管理を行う。この層により、上位側はディスクブロックを意識せずに2次記憶にアクセスを行うことが出来る。複合オブジェクト管理層は POM の最上層である。複合オブジェクト管理層はさらにオブジェクト管理部、型情報管理部の2つの部分からなる。オブジェクト管理部は、オブジェクトの生成、削除、参照、更新、および OID の生成を行い、型情報管理部は、型情報の生成、削除、参照、および TID の生成を行う。OID からオブジェクト、TID から型情報への参照は、B^+-tree を介して行われる。

オブジェクトの生成は、以下のようにして行われる。ユーザは予め定義した型の TID を指定して、オブジェクトの生成を POM に依頼する。POM 内部では、まず型情報管理部が生成すべきオブジェクトのサイズを算出し、可変長エリア管理層、ディスクブロック管理層を用いて、新たなオブジェクトのための領域を2次記憶上に確保する。一方、オブジェクト管理部は OID を生成し、新たなオブジェクトの物理位置を OID をキーとした B^+-tree に格納し、この OID をユーザ側に返す。

オブジェクトの参照、更新などのアクセスは、以下のようにして行われる。POM 内部では、まずオブジェクト管理部が B^+-tree を用いてユーザが指定した OID からオブジェクトの物理位置を得て、また型情報管理部がそのオブジェクトの型情報をもとにオブジェクト内のアクセスサイズ、アクセス位置を算出する。可変長ブロック管理層、ディスクブロック管理層は、これらの情報をもとにして2次記憶へのアクセスを行う。

4 実現

Sun3 上で POM を実現した。実現には C 言語を用い、プログラム約 2200 行で実現している。また、実現には Marcus J. Ranum 氏の B^+-tree の2次記憶インターフェースをチューニングしたもの (約 2900 行) を用いている。

POM の上位アプリケーションとして、POM 上で UNIX 風の階層ファイルシステムを構築した。型定義の例を図 2に示す。本ファイルシステムは UNIX の内部構造を模して作られており、dir_entry、inode、block、indirect_block、inindirect_block、ininindirect_block のオブジェクトより構成されている。ファイルの基本操作プリミティブとして、UNIX のシステムコールと互換性のある open、close、lseek、read、write、unlink、mkdir、rmdir をサポートしている。複合オブジェクトの概念を用いて実現したため、これらのプリミティブの実現は、非常に見通しの良いものになった。

```
type direct_block = byte_sequence(1024);
type indirect_block = array {OID(direct_block);};
type inindirect_block = array {OID(indirect_block);};
type ininindirect_block = array {OID(inindirect_block);};
inode_type = tuple{
   int access_mode;
   long access_time;
   long modification_time;
   long creation_time;
   union switch (file_type) {
     case DIR:
       dir_type;
     case ORD:
       tuple {
         OID direct_block;
         OID direct_block;
                :
         OID indirect_block;
         OID inindirect_block;
         OID ininindirect_block;
       };
   };
};
dir_type = array {tuple {
      inode_type inode;
      byte_sequence_type filename;}};
```

図 2: UNIX 風ファイルシステム内各オブジェクトの型定義

5 まとめ

本稿では、XERO における永続オブジェクト管理システムの設計と実現について述べた。今回性能を測定した POM はプロトタイプであり、POM 中にキャッシュバッファを設けていないなど、高速化のための技法をほとんど取り入れていない。そのため全体に処理が遅いものとなっており、性能的な改善の余地が多くある。今後永続キャッシング技術 [Kato92] など様々な高速化の技法を取り入れ、処理速度の向上をはかる予定である。また、現在のプロトタイプは単一サイトでの実現であるため、今後分散化、異機種上での実現を行う予定である。

謝辞

本研究を進めるにあたり有益な討論をしていただいた益田研究室の皆様、および本研究をする機会を与えてくださった (株) 日立旭エレクトロニクスの皆様に感謝致します。

参考文献

[加藤 89] 加藤, 猪原, 脇田, 益田 “データベース処理を指向した分散オペレーティング・システム XERO の構想,” 電子情報通信学会コンピュータシステム研究会 CPSY-89-29, 1989.

[加藤 91] 加藤, 猪原, 成田, 千葉, 益田 “無指向的オペレーティングシステム XERO の設計,” 第3回コンピュータシステムシンポジウム予稿集, pp.1-9, 情報処理学会, March 1991.

[Kato92] K. Kato and T. Masuda “Persistent caching: An implementation technique for complex objects with object identity,” IEEE Transactions on Software Engineering, 1992. To appear.

[成田 90] 成田, 加藤, 益田 “データベース指向 OS XERO における複合オブジェクト管理,” 情報処理学会第 40 回全国大会講演論文集, 1990.