ユーザレベルにおける分散共有メモリの実現

6C－3

似鳥 圭史　柳川 和久　芦原 評　清水 謙多郎

電気通信大学 情報工学科

1　はじめに

分散共有メモリは、主記憶を物理的に共有しないマルチプロセッサシステム、分散システムにおいて仮想的に共有メモリの機構を実現する方式である。その実現レベルについてはハードウェアレベル、オペレーティングシステム (OS) レベル、プログラミング言語レベル (ユーザレベル) の 3 つのレベルがある。

分散共有メモリをプログラミング言語レベルのオブジェクトを単位として実現する方式は、ページを単位としたOS レベルでの実現やハードウェアレベルでの実現に比べ、ユーザが共有のための操作を明示することが必要となるが、アプリケーションやシステム構成に応じた最適化を柔軟に行なうことができ、異種分散環境への適応も容易となる。

分散システム上で稼働しているプログラミング言語のオブジェクトを単位とする分散共有メモリは多数提案されている [1] が、本研究ではユーザが共有オブジェクトのアクセスを明示的に行ない、ユーザが一貫性を柔軟に指定できるようにしている点に特徴がある。類似の研究に [2] があるが、[2] が強い一貫性を採用しているのに対し、本研究ではシステムが提供するライブラリ群を使ってより柔軟な一貫性制御を可能にしている。

本稿ではプログラミング言語 (C++) のオブジェクトを単位とした、ユーザレベルの分散共有メモリの実現について述べる。実現にあたりノード間のデータ転送には Sun ONC/RPC の XDR を使用し、異種分散環境への適応を図った。現在、アーキテクチャの異なる 4 機種のワークステーションから構成される分散システム上で稼働している。

2　基本設計

本システムはユーザが分散を意識せずに、オブジェクトの共有を行なうことができる。オブジェクトは文字列名を持ち、分散システムにおける大域的な識別子として利用される。ユーザがオブジェクトにアクセスする際には必ずローカルなコピー (レプリカ) が作られる。各レプリカはそのレプリカを利用するプロセスのアドレス空間に割り当てられ、局所的なキャッシュとして働く。オブジェクトを共有する場合は、そのための基本操作を実行する。基本操作を明示することによって、ソフトウェアキャッシュ[3] 同様、柔軟性の高い一貫性制御が実現される。図 1はオブジェクトの共有の様子を表したものである。

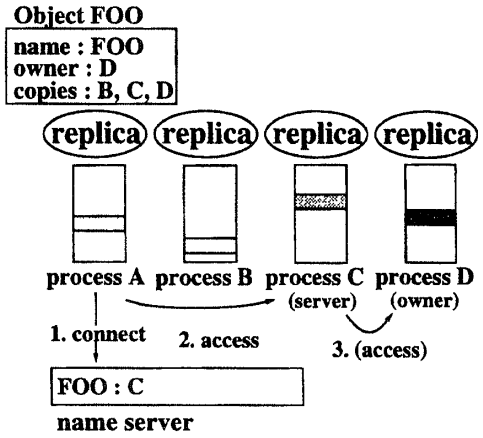

図 1: オブジェクトの共有

2.1　基本操作

共有操作に関するインタフェースを表 1に示す。これらの操作は共有オブジェクトのレプリカの基本クラスのメンバ関数として提供される。そのほか、並行性制御のための操作としてオブジェクトのロック、アンロック操作などがある。

表 1: インタフェース一覧

connect()	接続
access()	最新バージョンの取得
sync()	他のレプリカの更新
sync1()	封鎖しない sync()
invalidate()	他のレプリカの無効化
update()	指定レプリカの更新

Design and Implementation of the User-Level Distributed Shared Memory
Keishi Nitadori, Kazuhisa Yanagawa, Hyo Ashihara, Kentaro Shimizu
The University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182, Japan

2.2 実装

表1の操作インタフェースはC++プログラムから利用されるライブラリとして提供され、共有オブジェクトのレプリカとともにユーザのプログラムにリンクされる。

ユーザは共有オブジェクトの機能を利用する場合、connect() を実行する。最初にオブジェクトを使用するプロセス (サーバ) は、connect() により名前をネームサーバに登録する。以後同じオブジェクトにアクセスするプロセス (クライアント) は、指定した名前からネームサーバに問い合わせ、サーバの位置を求める。

サーバは、オブジェクトの一貫性制御に関する操作を行なう。サーバはレプリカを保持する他のプロセスのリスト (コピーリスト) と最新バージョンの保持者 (オーナ) の名前を持ち、各コピーに変更内容を通知する。

最新バージョンをアクセスするには access() を実行する。クライアントでオブジェクトの共有操作が呼び出されるとメッセージがサーバに送られる。

access() の要求メッセージを受けると、オーナから取得した最新バージョンを返し、要求クライアントをコピーリストに登録する。sync() の場合は各コピーへの通知と、レプリカの更新の両方が実行される。

メッセージを受けとるとユーザがあらかじめ定義していたハンドラが呼び出される。ハンドラは基本クラスの仮想メンバ関数であり、整数の識別子、送り手の名前、パラメータのストリーム形式、オブジェクトのストリーム形式を引数にとる。これによって sync や update によってレプリカの変更が通知された時、あるいは invalidate によって無効化された時の操作をユーザが指定することができる。また、非同期事象の通知のための機構を用意し、そのための処理もユーザがハンドラとして指定できるようにしている。

3 一貫性制御

ユーザが共有のための操作を明示するプログラミングスタイルにおいては、従来より柔軟な一貫性制御を行なうことができる。一般に、読み出し時に常に最新の内容が得られる方式が望ましいが、その実現には大きなコストを要する。一方アプリケーションによっては、最新の情報が必ずしも必要ない場合も多い。また、オブジェクトにバージョン番号を持たせることで [4]、ユーザ定義の因果関係とマルチバージョン管理に基づいたデータ一貫性制御方式への適用も可能である。

ユーザは sync, update あるいは invalidate のためのハンドラを再定義することによって、これらの事象を一貫性をとるきっかけとして利用できる。

4 使用例

図2に示すように、共有オブジェクトを使用するには、単に基本クラス Shared_Obj を継承し、オブジェクトからバイト列への変換を行なう関数 pack() とその逆の変換を行なう unpack() を定義すればよい。レプリカクラスの protected メンバ packed_obj は XDR ストリームのために提供され、pack(), unpack() は、この packed_obj に対する入出力を定義する。

```
class Shared_Counter:Shared_Obj{
  pack(){
   packed_obj.clear();
   packed_obj << c;
  }
  unpack(){
   packed_obj.open();
   packed_obj >> c;
  }
 public:
  int val;
  void operator ++() {
   lock();  access();
   val++;
   sync();  unlock();
  }
}

main(){
  Shared_Counter c;
  c.connect("Counter");  c.access();
  while (c.val < 10)
    if (f(c.val)) c++; else cout << c;
}
```

図 2: 使用例

参考文献

[1] A. S. Tanenbaum, *Distributed Operating Systems*, Prentice Hall, 1995.

[2] K. L. Johnson, M. F. Kaashoek and D. A. Wallach, "CRL: High-Performance All-Software Distributed Shared Memory", *Proceedings of the 15th ACM Operating Systems Principles*, pp.213-228, 1995.

[3] M. C. Carlisle and A. Rogers, "Software Caching and Computation Migration in Olden", *Proceedings of the 5th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp.29-38, 1995.

[4] 栗山 穣, 芦原 評, 清水 謙多郎, 因果関係とマルチバージョン管理に基づいたデータ一貫性制御方式, 情報処理学会第52回全国大会予稿集, 3M-6, 1996.