因果関係とマルチバージョン管理に基づくデーター貫性制御方式

3M－6

栗山 穣　芦原 評　清水 謙多郎
電気通信大学情報工学科

1. はじめに

分散システムにおける性能および信頼性向上のための手段として，オブジェクトのレプリケーションがある．レプリケーションで問題となるのがレプリカ間のデーター貫性の維持である．本稿ではデーター貫性制御方式の一つとしてユーザ定義の因果関係とマルチバージョン管理に基づくデーター貫性制御方式 MVC(Multi-Version Consistency)の提案を行う．

2. 基本方式

本方式は，基本的には，変更内容が最終的にコミットされるということだけを規定した slow memory[1]を保証し，ユーザが必要に応じて操作の因果関係，すなわち実行すべき操作の順序関係を明示し，一貫性を制御することで，アプリケーションに柔軟かつ効率的に対応することを目指す．順序関係の指定は操作対象となるバージョンの指定によって行い，ある操作を他の操作より後で実行することが論理的に必要である場合，先に行われた操作で得られたバージョンを次の操作の入力パラメータとする．MVC では以下のような命令群が定義される．

- VRead(*obj*,*buf*,v_i)　return(v_o)
  オブジェクト obj の内容をバッファ buf に読み出す．操作は，与えられたバージョン v_i 以降のオブジェクトに対して行われることが保証され，読出した結果のバージョンは v_o で返される．
- VWrite(*obj*,*buf*,v_i)　return(v_o)
  バッファ buf の内容をオブジェクト obj に書き込む．操作は，与えられたバージョン v_i 以降のオブジェクトに対して行われることが保証され，書込んだ結果のバージョンは v_o で返される．

オブジェクトに対する更新は上書きではなく，新しいバージョンを作成する形で行われる．レプリカのバージョン番号は，そのレプリカの存在するノードの番号と，そのバージョンが作成された時刻（タイムスタンプ）の対によって大域的に一意に定義される．あるバージョンを更新したバージョンを元のバージョン（親）の子という．あるバージョンに対する直接の子は複数存在しうるが，それらのうち，そのノードで最初に作成されたものを第一子という．最新版とはそのバージョンに対する子がその時点で存在しないバージョンである．最新版は複数存在する可能性があるが，あるバージョンを始点として連続する第一子の最新版は1つに定まる．これを，始点としたバージョンに対する第一最新版という．ノードに存在する他のどのバージョンよりも古いバージョンがただ1つ存在し，これを現バージョンという．よって親子関係に基づき，現バージョンを根，最新版を葉とする木がノードごとに構成される．あるバージョンの子が複数あるとき，投票によって1つが選ばれる．あるバージョンに対して，システム全体で唯一の子であることが確定することをコミットといい，それ以外の子孫およびその子孫は消去される．すでに消去されたバージョンの番号は消去リストとして記録され，障害からの回復などに用いる．

3. アルゴリズム

分散システム内の1つのノード（要求ノード）が操作を実行すると，その処理要求はレプリカを保持するノードの1つ（選択ノード）に送られる．読み出し操作の処理は，要求ノードと選択ノードの間のインタラクションだけで完結するが，書き込み操作の処理においては，選択ノード以外のレプリカを持つノードとも一貫性に関して調整する必要がある．書き込み操作の処理における選択ノードは，その処理の調停者となり，レプリカを保持する他のノードとバックグラウンドで更新処理を行う．

VRead, VWrite 命令の動作を示す．以下では，説明の便宜のため，バージョン v に対応するバージョン実体を V のように示す．

- VRead(obj,buf,v)

1. V が存在しない場合，V が作成されるまで待つ．
2. V が既に存在する場合は，V に対する第一最新版 U を読みだす．

- VWrite(obj,buf,v)

1. タイムスタンプを1増加させ，V の第一子 W を新たに作成する．

"A Method for Data Consistency Control Using User Defined Causality and Multiple Version Management"
Minoru Kuriyama, Hyo Ashihara, Kentaro Shimizu
The University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182, Japan

2. 要求ノードにwを返す.
3. 他のすべてのレプリカに対し更新メッセージを送信する.

レプリカを保持する各ノードにおいて,

4. 更新メッセージを受信する.
5. Vが存在しない場合はVが作成されるまで待つ.
6. Vの状態に応じて,

(i) Vが最新である場合. WをVの第一子として作成し, 選択ノードに対しWに対する投票メッセージを送信する.

(ii) Vに対する第一子W'がすでに存在する場合. Wを第一子でない子として作成し, 選択ノードに W'に対する投票メッセージを送信する.

選択ノードにおいて,

7. 他のすべてのレプリカからの投票メッセージを受信する.
8. 競合する書込みの中で最大数の票を獲得したバージョン(ここでは X とする)を決定しコミットする. コミットに際しては, コミットされたバージョン以外の(投票に敗れた)Vの子孫をすべて消去し消去リストに加える.
9. W に対する投票メッセージを送ってきた各レプリカに対してXに対するコミットメッセージを送信する. W'に対して投票してきたノードにはW'を生成したノードから送られる.

レプリカを保持する各ノードにおいて

10. コミットメッセージを受信する.
11. Xをコミットする.

4. 分散共有メモリへの適用

本研究では MVC をメモリ一貫性制御方式に用いた分散共有メモリ(DSM)[2]を, Ethernet により接続された複数の Unix ワークステーションからなる LAN 上で開発している.

従来のページフォルトに基づいたDSMと異なり, ユーザが明示する共有メモリへのアクセス命令をもとに一貫性制御を行うため, DSMをユーザレベルのライブラリとして実現することができた(図 1). また, ノード間の通信は MPI 通信ライブラリ[3]を介して行う. これによりDSMをハードウェアやOSに依存しない設計とすることを目指す.

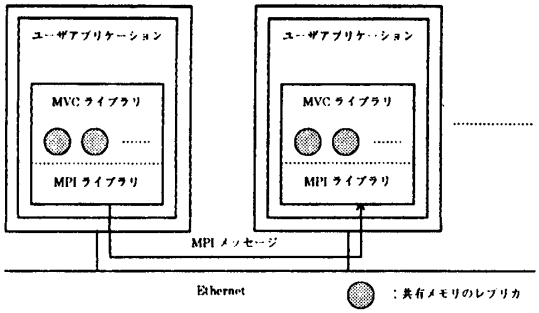

図 1 システム構成

5. 従来の研究との比較

ユーザが因果関係を指定する類似の方法は[4][5]で提案されている. [4]の方式では, 各レプリカは同時に 1 つのバージョンしか保持できず, 複数の書込みが並列に行われた場合, ユーザが特別な操作を実行しない限りレプリカ間の一貫性は保たれない. [5]の方式は, バージョンの受け渡しがユーザからシステムだけに限られているため, 数値計算などあらかじめ操作の順序関係が定まっている場合には有効だが, MVC のように動的に因果関係を変更することはできない.

6. まとめ

本稿では MVC の基本方式と分散共有メモリへの適用についてのべた. MVCの特徴をまとめると以下のようになる.

- 大域時間や集中制御ノードを必要としない.
- 書込みは非同期に行われ, 待ちがない.
- 高価な大域的同期を必要としない.
- 異なるバージョンに対する同時読み書きを許し, 並列性を向上させることができる.

参考文献

[1]P.W.Hutto and M.Ahamad.h : Slow Memory:Weakening Consistency to Enhance Concurrency in Distributed Shared Memories, *Proc. 10th Int. Conf. Distributed Computing Systems*, pp.302-311, 1990.

[2]B.Nitzberg and V.Lo : Distibured Shared Memory:A Survey of Issues and Algorithms, *IEEE Computer, Vol.24*, No.8, pp.52-60, 1991.

[3]Message Passing Interface Forum : MPI:A Message-Passing Interface Standard, 1994.

[4]R.Ladin, B.Liskov, L.Shrira, and S.Ghemawat : Providing High Availability Using Lazy Replication, *ACM Trans. Comput. Systems, Vol.10*, No.4, pp.360-391, 1992.

[5]M.J.Feeley and H.M.Levy : Distributed Shared Memory with Versioned Objects, *Proc. OOPSLA '92*, pp.247-262, 1992.