システムソフトウェアと
オペレーティング・システム 71－8
マルチメディア通信と分散処理 73－8
（1995．11．30）

分散オペレーティングシステムDM-2における メモリ管理部の実現

青木 秀貴† 篠原 拓嗣† 藤川 賢治† 大久保 英嗣†† 津田 孝夫†

† 京都大学工学部情報工学教室
†† 立命館大学理工学部情報学科

分散OS DM-2では、write-invalidate方式による厳密なメモリ整合性制御に加えて、delayed-update方式による弱いメモリ整合性制御を提供する。delayed-update方式は、不可分な操作によって扱う必要のあるデータ(クリティカルデータ)に対するアクセスの並列性を高めるものである。ユーザは書き込みロックを用いてプログラムを記述する必要があるが、一般にクリティカルデータを扱う際にはロック機構の使用が不可欠であり、このような場合には、ユーザの負担をほとんど増大させることなく、高いアクセス効率を得ることができる。本論文ではまた、write-invalidate方式の分散仮想記憶の性能評価の結果も示す。

An Implementation of Memory Manager in the Distributed Operating System DM-2

Hidetaka Aoki†, Takuji Shinohara†, Kenji Fujikawa†, Eiji Okubo††, Takao Tsuda†

† Department of Information Science, Kyoto University
†† Department of Computer Science, Ritsumeikan University

The distributed operating system DM-2 adopts a weaker memory consistency control scheme named *delayed-update* method in addition to the strict memory consistency control scheme based on write-invalidate method. The delayed-update method enables parallel accesses to critical data, which must be handled with atomic operations. Although this method requires write-locks in programs, it enables efficient accesses to critical data with little extra efforts because it is necessary to use locking primitives in dealing with such data in general. In this paper, we also show the result of the performance evaluation of the distributed virtual memory controlled by the write-invalidate method.

1 はじめに

我々が開発中の分散OS DM-2では、分散仮想記憶と呼ぶ方式によってメモリ資源の位置透過性を実現している。分散仮想記憶とは、ネットワーク上に広がる仮想的な単一のアドレス空間である。それによって各サイトが同じアドレスにアクセスすることが可能となった時、各サイトでそのアクセスがどのような順番で観察されるかが問題となってくる。このような問題は共有メモリを持ったマルチプロセッサ環境においても存在し、この分野において、これまでにいくつかのメモリ整合性モデルが提案されている[1]。

DM-2の前身となったDM-1では、write-invalidate方式のメモリ整合性制御を行うことによって、厳密な整合性を保証していた[2]。しかしwrite-invalidate方式による強い整合性制御では、メモリアクセスに要する待ち時間が長くなり、効率上不利となる場合も多い。

そこでDM-2では、write-invalidate方式の強いメ

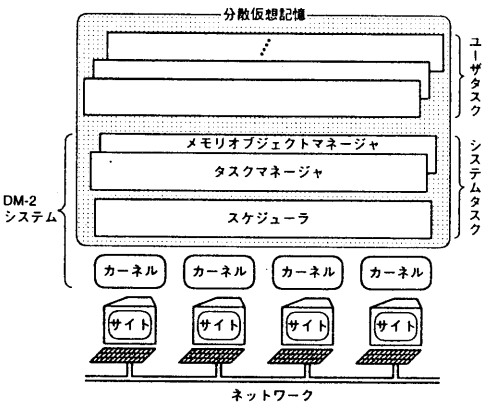

図 1: DM-2 の構成

モリ整合性制御に加えて、弱いメモリ整合性制御も提供する。ユーザはメモリオブジェクト毎にメモリ整合性制御方式を選択することができ、用途に応じた整合性制御を行うことで、効率のよいプログラムを記述することが可能になる。

本稿では、DM-2で採用する弱いメモリ整合性制御方式として、delayed-update 方式を提案する。delayed-update方式による整合性制御は書き込みロックの併用を前提としたものであり、本方式を利用することで、不可分な操作によって扱う必要のあるデータ (クリティカルデータ) に対する並列的なアクセスが可能になる。本方式を利用する際には、ユーザは書き込みロックを用いてプログラムを記述しなければならないが、一般にクリティカルデータを扱う際にはロック機構の併用が不可欠であり、このような場合には、本方式を用いてもユーザの負担はほとんど増大しない。

以下、まず 2 章で DM-2 と分散仮想記憶について説明する。次に 3 章で delayed-update 方式のメモリ整合性制御について説明し、4 章では現在実装されている write-invalidate 方式分散仮想記憶の性能評価を行う。

2 DM-2 と分散仮想記憶

我々は、分散仮想記憶に基づくオペレーティングシステム DM-2 の開発を行っている [3]。DM-2 が対象とするのは、数台から十数台程度の同一アーキテクチャの計算機をネットワークで接続したシステムである。図 1 に DM-2 の構成を示す。

DM-2 はマイクロカーネル方式に基づいて構築されており、各サイトで動作する分散カーネルは、OS のごく基本的な機能のみを提供する。分散仮想記憶はカーネルで実現されているため、OS のその他の機能を実現するサーバプロセス (DM-2 ではシステムタスクと呼ぶ) も、分散仮想記憶を利用することが可能となっている。

2.1 分散仮想記憶

分散仮想記憶とは、ネットワークで接続されたシステム上に存在する唯一の仮想的なアドレス空間である。システム中のメモリ資源 (主記憶および二次記憶) はすべてこの唯一の仮想アドレス空間上にマッピングされ、統一的に管理される。分散仮想記憶へのアクセスは、その実体の存在する物理的位置を意識することなく、任意のサイトから一様に行うことができる。

実行されるコードも分散仮想記憶上に配置されるため、スレッドの実行位置に関する透過性が達成され、サイトを越えた制御の移動も容易に実現できる。したがって、各サイトの負荷やネットワークの負荷を考慮に入れたスレッドの分配を行うことにより、システムの効率的な利用が可能となる [4]。

DM-2 では、CPU のページング機構を利用することにより、ページ単位で分散仮想記憶の管理を行っている。実際に分散仮想記憶を実現しているのは、カーネル中のメモリ管理部と呼ばれる部分であり、具体的にはメモリ資源の管理およびサイト間の整合性制御を行っている。

2.2 複数の整合性制御方式のサポート

分散仮想記憶ではすべてのサイトが同じアドレスにアクセスすることが可能であり、ページに対する並列なアクセスを可能とするため、ページの複製が複数のサイトに存在することを許している。ここで、分散仮想記憶がシステムに唯一のアドレス空間をあたえるものであることを考えると、各サイトのページ複製はすべて同一内容を持つべきであり、複製間の強い整合性制御が必要ということになる。

DM-2 では、DM-1 と同様、write-invalidate 方式の強い整合性制御を行う分散仮想記憶を提供している。write-invalidate 方式の分散仮想記憶では、あるページの複製の内容が、その時点でそのページにアクセスしているすべてのサイトで同一であることが保証されており、分散共有メモリにおける Atomic Consistency (AC) [1] に従っているといえる。プログラミングモデルはわかりやすいものとなり、ユーザは従来のプログ

ラミングモデルの自然な拡張によって、複数のプロセッサを使用できる。

しかし、このような強いメモリ整合性制御を行った場合、メモリアクセスに要する待ち時間が長くなり、効率上不利となる場合も多い。例えば、必ずしも最新のデータがなくとも十分な動作が見込めるような場合や、あるいはプログラマがもともと同期を意識してプログラムを記述しているような場合などである。

そこで DM-2 では、write-invalidate 方式を用いた強いメモリ整合性制御に加えて、delayed-update 方式による弱いメモリ整合性制御も提供し、用途に応じたメモリ整合性制御方式をプログラマが選択できるようにする。

3 delayed-update 方式

一般に、不可分な操作によって扱う必要のあるデータ (以降、**クリティカルデータ**と呼ぶ) は、書き込みロックおよび読み出しロックを用いて排他制御を行い、クリティカルセクション内でのみそのデータを扱うことで、一貫性が保たれるようにする。書き込みロックおよび読み出しロックは、次のような働きを持つものである。

- クリティカルデータに書き込みを行う際にはまず書き込みロックを確保し、書き込みが終了した時点で書き込みロックを解放する。書き込みロックは、同時には一つのスレッドしか確保することができない。これによって、ある時点にそのデータを変更できるスレッドは唯一となり、書き込みにおけるデータの一貫性を保つことが可能となる。
- クリティカルデータを読み出す際にはまず読み出しロックを確保し、読み出しが終了した時点で読み出しロックを解放する。読み出しロックは同時に複数のスレッドで確保することができ、読み出しロックをかけている間の読み出しでは、データの一貫性が保証される。

分散環境では、次のようにすることで、クリティカルデータへのアクセスの並列性を高めることが可能である。

- あるサイトで書き込みロックがかけられている間も、他サイトでは一貫性の保たれている古いデータを読み出すことを許す。
- あるサイトで読み出しロックがかけられている間も、他サイトでの書き込みを許す。ただしその書き込みにより、読み出しロックをかけているサイトのデータを壊すようなことがあってはならない。

しかしこれらの並列アクセス手法は、write-invalidate 方式のメモリ整合性制御では実現することができない。

そこで、上記の並列アクセスを可能にするため、delayed-update 方式のメモリ整合性制御を提案する。delayed-update 方式は write-invalidate 方式よりも弱いメモリ整合性制御方式であり、利用の際には、ロック機構および動作制御用のカーネルコール (カーネルが上位層に対して提供するインタフェース) の使用を必要とする。

3.1 前提

本方式を利用する際には、扱うデータがクリティカルデータであるかどうかにかかわらず、原則としてネットワークワイドに機能する書き込みロックを用いてプログラムを記述しなければならない。ネットワークワイドに機能する読み出しロックは必要ないが、クリティカルデータを扱う場合には、読み出し中のクリティカルデータを同一サイトの別スレッドが更新することのないよう、サイト内でのみ機能する局所的なロックなどを用いて適宜制御する必要がある。

また、ロックをかける単位が、整合性制御の単位と一致していなければならない。DM-2 ではページ単位で整合性制御を行っているため、ページ単位でロックをかけることになる。そのため、ページ内で false sharing が起こらないようにしなければならない。

なお、ユーザが次のことを守っているものとして話を進める。

- データに対して書き込みを行う時には、必ず書き込みロックをかける。
- 書き込みロックを解放する直前で、必ず transport カーネルコール (後述) を発行する。

3.2 整合性制御

DM-2 にあわせて、以下ではページ単位の整合性制御について説明する。

delayed-update 方式の具体的な整合性制御は、ページの複製に updated, dirty, old, unused の四つの属性をあたえるとともに、動作制御用のカーネルコール 'transport' および 'update' を用意することで行う。ページ複製の各属性の意味は、次の通りである。

1. **updated 属性**
   その複製には、一貫性のとれたデータのうち、最

新のものが格納されている。この属性の複製に対しては、読み出しは自由に行うことができるが、書き込みを行おうとするとページフォールトが発生し、dirty 属性へと遷移する。

また、他サイトよりページが転送されてきた場合には、old 属性へと遷移する。その際、それまでのページ内容を old 属性における表バンクとして使い、転送されてきた内容は裏バンクに書き込む。

2. **dirty 属性**

その複製は、一貫性のとれたデータの最新のものに対して何らかの書き込みを行ったものであり、一般にその内容は一貫性がとれていない。この属性の複製に対しては、読み出し・書き込みともに自由に行うことができる。

この属性に遷移した時には、ユーザは既に書き込みロックをかけているはずであり、この属性の複製はシステム内に高々一つしか存在しない。本属性の複製に対して一連の書き込みが行われデータの一貫性がとれた時点で、ユーザは transport カーネルコールを発行し、その後で書き込みロックの解放を行わなければならない。transport カーネルコールを受けると、dirty 属性の複製は updated 属性へと遷移するとともに、自身の内容を他サイトに転送する。

3. **old 属性**

この属性は、主記憶ないし二次記憶のページ枠を二つ消費し、表バンク・裏バンクとして使用する。表バンクには一貫性のとれたデータのうち最新ではないものが格納されており、読み出し専用として、old 属性である間はその内容が変化しないことが保証される。それに対し、裏バンクには一貫性のとれたデータの最新のものが格納されており、他サイトからページが転送される毎に、その内容は裏バンクに書き込まれる。

この属性の複製に対しては、ユーザは自由に読み出しを行うことができ、実際の読み出しは表バンクから行う。裏バンクに格納された新しいデータを読みたい場合には、ユーザは update カーネルコールを発行する。update カーネルコールを受けると、裏バンクをページ内容として updated 属性へと遷移し、表バンクは解放する。ただし、裏バンクにページ転送を受けている間は、update カーネルコールが発行されてもすぐには updated 属性に遷移せず、ページ転送が終わってから遷移する。

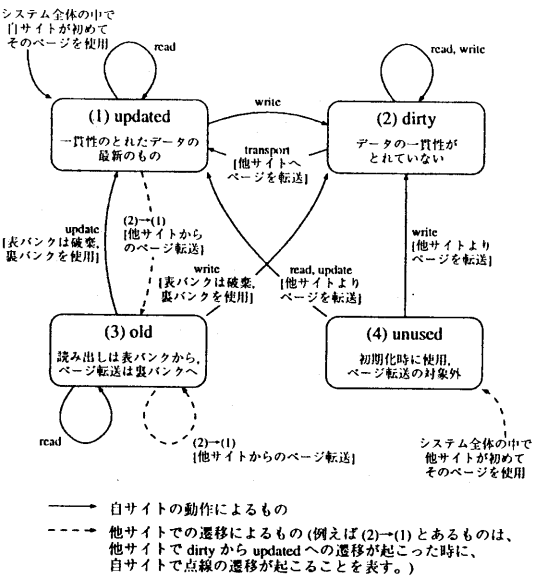

図 2: delayed-update 方式における複製の属性の遷移

一方、この属性の複製に対して書き込みを行おうとするとページフォールトが発生し、裏バンクをページ内容として dirty 属性へと遷移し、表バンクは解放する。

4. **unused 属性**

主に初期化時に使用される属性で、この属性の複製を持つサイトは、transport カーネルコールによるページ転送の対象外となる。あるページが全サイトを通じて初めて使われる時に、他のサイトではページの属性をこの unused 属性に設定する。ただし、unused 属性のページは実体が作られない。

unused 属性に対してアクセスを行おうとするとページフォールトが発生し、そのページを最初に使ったサイトからページ転送を受けた上で、読み出しアクセスなら updated 属性に、書き込みアクセスなら dirty 属性に遷移する。また、自サイトで update カーネルコールが発行された場合には、読み出しアクセスの場合と同様に、そのページを最初に使ったサイトからページ転送を受けた上で、updated 属性へと遷移する。

これらの状態の遷移を、図 2 に示す。

transport カーネルコールおよび update カーネルコールを発行する際には、対象とするページを指定する。transport カーネルコールは、発行サイトにおける

指定ページの複製がdirty属性の場合にのみ機能する。一方updateカーネルコールは、old属性かupdate属性の場合にのみ機能する。

なお、あるページを使用している(unused属性以外の複製を持つ)サイトは、どのサイトでそのページが使用されているかを把握している。transportカーネルコール時のページ転送は、その情報を用いてマルチキャストによって行う。

3.3 特徴

この delayed-update 方式によるメモリ整合性制御には、次のような特徴がある。

1. クリティカルデータを扱う時、write-invalidate方式ではロックによる待ちや通信による待ちが多くなる。本方式では書き込みロックの併用を前提とし、書き込み処理と読み出し処理、およびページ転送と読み出し処理をオーバラップさせることで、この問題を解決している。

2. 本方式では、ある時点でアクセス中のページ複製の内容が、一般に全サイトでは同じでない。その結果この方式は、分散仮想記憶の提供する「唯一のアドレス空間」の考えに、厳密には従わないものとなっている。

3. 書き込みロックを使うことで、同一ページ(同一データ)に対する書き込みが逐次化され、あるページの変更履歴は唯一となる。また書き込みロックを使っている限り、どのサイトでも複製の内容が時間的に逆行することはない。これらにより、書き込みロックを併用した場合には、この方式を用いたプログラムの実行で Sequential Consistency (SC) [5] に従った結果を得ることができる。

4. 書き込みロックを使っている場合には、あるページを利用しているすべてのサイトに、そのページの十分に新しい複製が存在している。具体的には、現在書き込みを行っていないサイトには、一貫性のとれたデータの最新のものが存在し、また書き込みを行っているサイトには、一貫性はとれていないがより新しい更新中のデータが存在する。そのため、障害発生時のデータの回復が容易である。

5. 書き込みロックを用いるとともに、読み出し中のold属性の複製に対してupdateカーネルコールが発行されることがなければ、データに対するアクセスは不可分であるとみなせる。よってその場合には、クリティカルデータを扱うことが可能である。一般にクリティカルデータを扱う際にはロック機構が不可欠であり、このような場合には、本方式を用いてもユーザの負担はほとんど増大しない。

6. 書き込みロックを使わない場合には、データの一貫性およびサイト間の整合性は保証されない。例えば、書き込みロックを行わず、さらにtransportカーネルコールもupdateカーネルコールも使用しなければ、各サイトの複製は完全に他サイトから独立なものとなる。

本方式を write-invalidate 方式と比べると、アクセス効率は向上しているが、old属性でページ枠を二つ必要とするため、メモリ資源の利用効率は低下している。

delayed-update方式の分散仮想記憶では、ある一つのサイトでは頻繁に書き込みが行われるが、他サイトではほとんどアクセスを受けないようなデータを扱う場合には問題が生じる。なぜなら、書き込みが終了する度に、他サイトではまったく使用されることのないページを転送していたのでは、ネットワークに余計な負荷がかかるためである。また、ページ転送を受ける側のサイトでは、受信バッファとして使われるold属性の裏バンクを、使う見込みがなくても常に主記憶上に確保しておくことになるため、主記憶の利用効率が悪くなる。これらの問題の解決策として、あるサイトではそのページをしばらくの間使用しないことがわかった時点で、その複製をunused属性にするプリミティブを用意することが考えられる。

データを読み出すスレッドが一つのサイトに複数存在する場合にも、注意が必要である。なぜなら、各スレッドは小まめに読み出しロックを解放しているつもりでも、サイト全体で見るとまったく読み出しロックが解放されない状況が起こり得るからである。この状況では、updateカーネルコールによってデータを新しいものに切り替える機会を失い、最初に読み出しロックを確保した時点の古いデータを読み続けることになる。このような状況で得られる結果も確かにSequential Consistencyに従ってはいるが、このような状況が発生する危険を、プログラマは認識する必要があろう。

3.4 適用範囲

delayed-update方式がその効力を発揮するのは、頻繁に更新される共有クリティカルデータが、その更新と同程度あるいはそれ以上の頻度で参照される場合である。

表 1: write-invalidate 方式分散仮想記憶の性能

処理	256 回の時間	1 回あたり
無効化を伴う処理	2.02 秒	7.89 ミリ秒
ページ転送を伴う処理	5.18 秒	20.2 ミリ秒

また、たとえ内容が少々古くても、待ち時間なしですぐにデータを参照することが重要な場合にも、本方式は有効である。

その他、「一つあるいは複数のスレッドがページの異なる領域に次々とデータを書き込んでいき、他のスレッド (一つでも複数でも可) がその書き込まれたデータを参照する」という状況 (例えば、あるサイトでページの頭からデータを書き込んでいき、それを他のサイトで順に読み出していくパイプライン処理など) でも有効である。

4 write-invalidate 方式分散仮想記憶の性能評価

我々は現在、IBM-PC 互換機 (搭載 CPU は Intel i486DX2/66MHz) をイーサネットで接続したシステム上に DM-2 を実装中である。現時点のメモリ管理部の実装では、2 サイトで動作する write-invalidate 方式の分散仮想記憶が利用可能である。

この 2 サイトで動作する write-invalidate 方式分散仮想記憶の性能を調べるため、通信を必要とする次の二種類の処理にかかる時間を測定した。

無効化を伴う処理 複製が複数サイトに存在するページに書き込もうとした場合の処理で、ページフォールトの発生から実際に書き込みを行うまでを指す。通信メッセージとしては、無効化要求および無効化完了通知が流れる。

ページ転送を伴う処理 自サイトに複製が存在しないページを読み出そうとした場合の処理で、ページフォールトの発生から実際に読み出しを行うまでを指す。通信メッセージとしては、ページ転送要求およびその返信 (4KB のページ転送を含む) が流れる。

実際の測定は、処理を 256 回繰り返すのに要した時間を計ることで行った。測定結果を表 1 に示す。

ページ転送を伴う処理の結果によると、他サイトにあるページを 1.54Mbps の速さで受け取ることができる。イーサネットの転送能力は最高で 10Mbps であり、種々のオーバヘッドを考えると、それほど悪い性能ではなかろう。

5 おわりに

本稿では、delayed-update 方式のメモリ整合性制御について述べた。delayed-update 方式を利用することで、不可分な操作によって扱う必要のあるデータに対する並列的なアクセスが可能になり、アクセス効率が向上する。

また、メモリ管理部で実現されている write-invalidate 方式の分散仮想記憶の性能評価も行った。

今後の課題としては、3 サイト以上で動作する write-invalidate 方式分散仮想記憶および delayed-update 方式分散仮想記憶を実装し、評価することが挙げられる。

謝辞

本研究を進めるにあたり貴重な御意見を頂いた、津田研究室の皆様に感謝いたします。

参考文献

[1] David Mosberger. "Memory Consistency Models", ACM Operating System Review, 27(1):18–26, January 1993.

[2] 篠原拓嗣, 藤川賢治, 大久保英嗣, 津田孝夫. "分散仮想記憶に基づくオペレーティングシステム DM-1 の構成", 情報処理学会研究会報告, 93-OS-61, August 1993.

[3] 篠原拓嗣. "分散仮想記憶に基づくオペレーティングシステム DM-2 の設計と実現", 京都大学大学院工学研究科情報工学専攻修士論文, 1995.

[4] 伊藤ちひろ, 篠原拓嗣, 藤川賢治, 大久保英嗣, 津田孝夫. "分散オペレーティングシステム DM-2 におけるスレッドディストリビュータの実現", 情報処理学会第 51 回全国大会 7L-1, 1995.

[5] Leslie Lamport. "How to make a multiprocessor computer that correctly executes multiprocess programs", IEEE Transactions on Computers, C-28(9):241–248, September 1979.