社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－MPS－60 (14)
2006／6／26

コモディティOSとメモリマップによるクラスタシステムの実装

金井 遵[†] 森 拓郎[†] 荒木健志[†]
田邊 昇[††] 中條拓伯[†] 並木美太郎[†]

本論文では，分散ファイルシステム (DFS) およびメモリマップトファイル機能を利用して OS に手を加えることなく分散共有メモリ (DSM) を実装し，カーネルに手を加えることができないコモディティOS 上でクラスタシステムを実現する方法を提案する．大容量バッファを持った高速なネットワークインターフェースである DIMMnet-2 を用い，Windows 上で複数の DIMMnet-2 の大容量バッファをまとめて一つの DFS および，DSM として利用するドライバとライブラリを設計，実装した．評価では実際に，DSM を用いていくつかの分散処理実験を行った．特に行列乗算による評価では，2 ノードの分散処理において 1.99 倍の性能向上が予測できた．

Implementation of PC Cluster System with Memory Mapped File by Commodity OS

Jun Kanai ,[†] Takuro Mori ,[†] Takeshi Araki ,[†]
Noboru Tanabe ,[††] Hironori Nakajo[†] and Mitaro Namiki [†]

This paper describes implemention of Distributed Shared Memory(DSM) by using Distributed File System(DFS) and Memory Mapped File without changing souce code of OS in order to implement PC Cluster System for a non-open source commodity OS. We have designed and implemented a DFS device driver and a DSM library by plural high-speed network interface cards named DIMMnet-2 with mass buffer for Microsoft Windows. As a result of matrix multiplication evaluation, up to 1.99 times higher performance has been gained by 2-nodes distributed parallel execution.

1. はじめに

近年，HPC (High Performance Computing) の分野において，多数の PC を相互に接続した PC クラスタシステムの躍進は目覚しい．現在のクラスタシステムの環境はほとんどが Unix をベースとしたものである．しかし，今や Microsoft Windows のシェアは 97%以上に達し，今後の分散処理においても重要な存在になり得ると考えられる．Windows 環境下での分散コンピューティングの可能性を示すことは，ソースコードがなく，OS のカーネルに手を加えることができないコモディティOS 環境での分散処理の可能性を示すことにもなる．

PC クラスタの躍進を背景に，安価にシステムを構築できる DIMM スロットへハードウェアを接続する HPC 用高速ネットワークインタフェース DIMMnet-2[3] が開発された．DIMMnet-2 は従来の汎用バスに接続するタイプの HPC 用 NIC に比べ 1/10 程度のアクセスレイテンシ，最上位レベルの帯域幅を実現している．また，DIMMnet-2 は，SO-DIMM による数百 MB～数 GB の大容量バッファを持ち，通信用バッファやデータ待避用領域として利用することが可能である．

しかし，DIMMnet-2 はシステムソフトウェア面では，Linux 用にコマンド発行のための低レベルなドライバがあるのみで，Windows をはじめとしたコモディティOS 環境下の分散処理環境が整っていない．そこで本稿では，DIMMnet-2 を用い，Windows 向けに DIMMnet-2 用デバイスドライバを設計および実装することで，OS に手を加えることなく，コモディティOS とメモリマップによってクラスタシステムを構築し，評価を行う．

2. DIMMnet-2 プログラミングの問題点

本章では，従来のクラスタシステムで用いられるデータ共有方法であるメッセージパッシングと分散共有メモリ (DSM) について述べ，DIMMnet-2 とコモディティOS 環境下での実現方法について考察する．

メッセージパッシングは，データ共有を行うために明示的にデータのやりとりを行う方法であり，現在，HPC において最も多く利用される分散データによるプログラミング方法である．MPI[1] をはじめとしたメッセージパッシングでは，一般的に性能が優れるという利点があるが，一方でデータの送受信について明示しなければならないため，プログラマにとって負担が大きい．メッセージパッシングを実現する場合，帯域や遅延が最重要視されるため，OS を介することによるオーバヘッドを極力減らす必要があり，ユーザランドで処理を行うのが望ましい．そのためには既存の Linux 用 DIMMnet-2 ドライバにおいては各バッファ (ウィンドウ) やレジスタをユーザ空間にマップした上で，DIMMnet-2 のウィンドウを用いた複雑な間接アクセス機構を意識しながら，明示的にコマンド発行を記述する必要があり，プログラミングの手間がかかる．

一方，分散共有メモリ方式 (DSM) は分散メモリ型マシンにおいて，実際に共有メモリがなくても共有メモリが存

† 東京農工大学
Tokyo University of Agriculture and Technology
†† (株) 東芝，研究開発センター
Corporate Research and Developpment Center, Toshiba

在するように見せかける技術である．DSM 方式では，明示的にデータのやりとりを記述する必要がないため，プログラミングが容易である一方で，メッセージパッシング型よりも性能が劣ることが多い．

従来の MMU を利用する DSM での問題は，今回のように OS に手を加えることが出来ない環境下で利用が難しいこと，ユーザモードのみで DSM を実現する場合の問題は MMU が利用できずオーバヘッドが発生することである．そこで，本稿ではメモリマップトファイルと DFS により，OS に手を加えることができない環境下で間接的に MMU を利用して DSM を実現する方法を提案する．

メモリマップトファイルは仮想アドレス空間にファイルをマップする方法であり，MMU を利用して実装される．DFS 上に共有データを置き，メモリマップトファイルを利用することで，OS に手を加えずに MMU を利用して，DSM の実現ができる．共有メモリを作成した場合に自動的に必要なページのみがロード，ダーティページのみがライトされるため，通信頻度の最適化や DSM 管理用ソフトウェアによるオーバヘッドの削減が可能である．また，各ノードでキャッシュを持ち，速度面で優位である．

また，本方式で DSM を実現する場合，ドライバレベルで DFS を実装するのみで良く比較的実装が容易であると考える．DIMMnet-2 においては大規模な SO-DIMM によるバッファを持ち，ネットワークを利用し，広帯域・低遅延な SO-DIMM 内のデータの送受信が可能である．このバッファを DFS や DSM の記憶領域とする DFS や DSM が本方式に適すると考える．

3. 本研究の目標

本研究では Windows とメモリマップおよび DIMMnet-2 を利用して，クラスタシステムを構築し，評価を行う．

そこで，まず DIMMnet-2 の大容量バッファを RAM ディスクのディスク領域として利用する分散ストレージドライバ AT-dRAM を開発し，DSM を実現し評価を行う．また，従来の主記憶の代替手段として，実用的な速度を実現できるかを検証する．次に，通信をユーザモードのみで完結し，高速なメッセージパッシングシステムを実現や，DIMMnet-2 の拡張機能の利用を可能にするためのドライバ MT-dNET を開発し，有用性について検証する．

4. クラスタシステム用ドライバの概要

前章で述べた考察をもとに，AT-dRAM および，MT-dNET の設計，実装について述べる．今回開発したシステムの全体構成を図 1 に示す．

AT-dRAM は，DIMMnet-2 の大容量バッファをディスク領域とする RAM ディスクドライバであり，DIMMnet-2 の煩雑な間接アクセス機構をプログラマに対して隠蔽するとともに，クラスタシステムで多く利用される DSM，DFS を実現する．AT-dRAM では，複数の DIMMnet-2 を一つのディスクとして仮想化し，リモートの DIMMnet-2 の大容量バッファへのアクセスを隠蔽する．これにより，通常のファイルアクセス手段により，ローカルとリモートの DIMMnet-2 大容量バッファへのアクセスが可能になる．

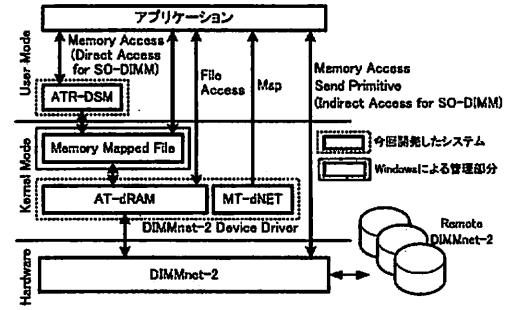

図 1　システム全体構成図

```
mat_a = (CAST)dn_malloc(MATSIZE);
mat_b = (CAST)dn_malloc(MATSIZE);
mat_c = (CAST)dn_malloc(MATSIZE);
//通常の主記憶利用と同じ方法
for (i = 0; i < SIZE; i++)
  for (j = 0; j < SIZE; j++)
    for (k = 0; k < SIZE; k++)
      (*mat_c)[i][j] += (*mat_a)[i][k]
                      * (*mat_b)[k][j];
```

図 2　行列乗算例 (AT-dRAM とメモリマップトファイル利用)

複数のノードにおいてドライバをロードすることで，DFS としての利用も可能である．また，従来の DIMMnet-2 プログラミングでは DIMMnet-2 の間接アクセス機構を意識する必要があったが，メモリマップトファイルの利用により，従来の主記憶を利用するアルゴリズムがそのまま適用可能になり，DIMMnet-2 プログラミングが容易となる．さらに，複数のノードから同一のファイルをマップすることにより，DSM としての利用が可能である．

分散処理を行う場合に，より実用的な DSM とするためには，共有メモリ領域の作成やバリア同期等の機能が必要になる．そのため，これらの機能をサポートするライブラリ ATR-DSM も同時に開発した．厳密な DSM とする場合，一貫性制御が必要になるが，今回の場合は性能を最重視し，ライブラリでの一貫性制御は行わず，明示的にデータのフラッシュ，更新を行う仕組みとした．

一方で，メッセージパッシングシステムの実装，ストライド命令をはじめとした各種拡張命令の利用時など，OS によるオーバヘッドを減らし，明示的に DIMMnet-2 を操作したい場合が存在する．そこで，応用プログラムからユーザモードのみで DIMMnet-2 の明示的な操作を可能にするドライバ MT-dNET を開発した．

MT-dNET では各種ウィンドウやレジスタをプロセスのユーザー空間へマップする機能を提供する．これにより大容量バッファへのアクセスや，リモートノードの DIMMnet-2 へのアクセスをユーザモードで完結することが可能になり，OS 呼出しによるオーバヘッドをなくし，通信の遅延を極力おさえることが可能である．

5. プログラミング例

5.1 ローカルでのプログラミング例

AT-dRAM と，ユーザモードライブラリを使い，行列の乗算を行う場合のプログラミング例を図 2 に示す．

例より，AT-dRAM では DIMMnet-2 の間接アクセス機構を意識することなく，従来とほぼ同じアルゴリズムが

適用可能であり，プログラミングが容易である．また，ファイルシステムキャッシュが利用可能なため，DIMMnet-2へのアクセスが最低限に抑えられ，性能向上が期待できる．

6. 評　価

6.1 OSによる各種オーバヘッド

AT-dRAMはストレージシステムという形で実装されているため，ドライバ呼出し等のコストが発生する．よって，ここでは下記式を用い，明示的にAPIを利用してファイルアクセスを行った場合のOSによるオーバヘッドを計算した．ここで，T_{os}はOSによるオーバヘッド，N_{call}はAPIによるOS呼出し回数の実測値，N_{size}は一度に転送するサイズの実測値，T_{TR}は一度の転送にかかる時間の実測値である．総時間とAPI呼び出し回数の差，$\Delta_{T_{all}}$，$\Delta_{N_{call}}$を求めることにより，1回のOS呼出しにかかるコストを計算している．

$$T_{all} = T_{os}N_{call} + N_{size}T_{TR} \quad (1)$$

$$T_{os} = \Delta_{T_{all}}/\Delta_{N_{call}} \quad (2)$$

続いて，メモリマップトファイルを用いてファイルアクセスを行った場合のOSのオーバヘッドT_{mmap}を計算した．読み込みの場合，ページ毎（4KB）にページフォールトが発生し，ストレージドライバが呼び出され，該当ページのデータがディスクから読み込まれる．

$$T_{mmap} = (T_{all} - T_{TR}N_{size})/N_{page} \quad (3)$$

ディスクからの読み込みにかかる時間は(1)式から算出可能であり，これを利用し，(3)式からメモリマップトファイルを利用した場合のページ単位のOSのオーバヘッドを計測した．T_{TR}は単位あたりのディスクからの読み込み時間の実測値を，N_{size}の実測値は読み込んだサイズの合計を，N_{page}はページ数の計算値をそれぞれ表す．

メモリマップトファイル利用時ののデータ書き込みは基本的にOSにより自動的に行われるが，明示的に書き込みを指示することもでき，ATR-DSMではこちらを主に利用する．この場合のオーバヘッドを(2)式により計算した．オーバヘッドをまとめたものを表1に示す．

メモリマップトファイルを利用した場合では，Read-FileAPIを利用した場合に比べ，オーバヘッドが大きくなっているが，メモリマップトファイルでは最初の一回のアクセスのみでこのオーバヘッドが発生するため，明示的に複数回ReadFileを呼ぶような場合に比べると，性能的に有利である．

6.2 DIMMnet-2 ホスト間転送速度

MT-dNETを用いて，OSを介さない場合のDIMMnet-2のSO-DIMMバッファとホストメモリ間の転送性能の計測を行った．結果を図3に示す．

現在，ハードウェアの不具合が若干あり，DIMMnet-2の各ウィンドウのキャッシュ属性はUncachedとして動作している．将来的にはWrite Window(SO-DIMM書込み用バッファ)はWriteCombineに，Prefetch Window(SO-DIMM読込み用バッファ)はCached☆で動作するようになり，大幅な性能向上が望める．

表1　ファイルアクセスのOSオーバヘッド

	Read[μ s]	Write[μ s]
ReadFile/WiteFileAPI	4.27	3.75
MemoryMappedFile	9.87	4.39

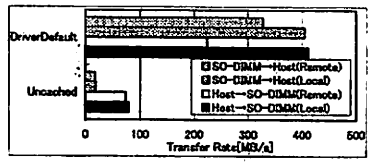

図3　DIMMnet-2 ホスト間転送速度（実測値）

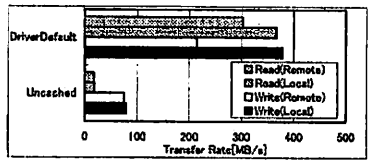

図4　ディスク性能（DriverDefaultは推定値）

6.3 分散ファイルシステム性能

キャッシュ属性Uncachedの場合のDFS性能とドライバを介さない場合のDIMMnet-2のSO-DIMMバッファとホストメモリ間の転送性能の実測値と下記式を用いて，DriverDefaultの場合のディスク性能T_{DD}を見積った．

$$T_{DD} = T_{os} + T_{DDD} \quad (4)$$

$$= (T_{all} - T_{DUN}) + T_{DDD} \quad (5)$$

ここでT_{os}は，OS呼出しによるオーバヘッドを，T_{DDD}は実測値でDriverDefaultでOSを介さない場合のSO-DIMMへの転送時間を，T_{all}は実測値でUncachedでOSを介した場合のファイルアクセス時間の総計を，T_{DUN}は実測値でUncachedでOSを介さない場合のSO-DIMMへの転送時間を表す．

結果を図4に示す．DriverDefaultでは，MT-dNETを利用する場合に比べ，ディスク化した場合にはファイルシステム管理や，OSによるオーバヘッドが原因となり，最大9%の性能低下になっているが，GbEでSMBを用いてディスク共有を行った場合の性能上限30[MB/s]と比べると最大12倍以上になり，DFSとして大幅な性能向上を達成できると考えられる．

6.4 分散処理による評価

AT-dRAMおよびATR-DSMを用いて，各種分散処理に関する評価を行った．なお前述したとおり，現在DIMMnet-2の各ウィンドウのキャッシュ属性がUncachedとして動作しており，ハードウェアの不具合の修正により性能が改善した場合の性能予測を下記式を用いて行った．

$$T_{all} = T_{os} + T_{barrier} + T_{tr} + T_{calc} \quad (6)$$

$$T_{os} = T_{mmap}N_{page} + T_{flush}N_{flush} \quad (7)$$

$$T_{tr} = \sum_{i=0}^{N} Tr_i Nr_i + \sum_{i=0}^{N} Tw_i Nw_i \quad (8)$$

$$T_{barrier} = T_{all} - max(T_{os_i} + T_{tr_i} + T_{calc_i}) \quad (9)$$

ここでディスクへの読み込み回数Nr_i，書き込み回数Nw_i，およびUncachedの場合の読み込み時間Tr_i，書き込み時間Tw_iは実測値である．現在，リモート転送に関して，ソフトウェアで再送制御を行っており，iが1以

☆ このキャッシュ属性の組み合わせをDriverDefaultとよぶ

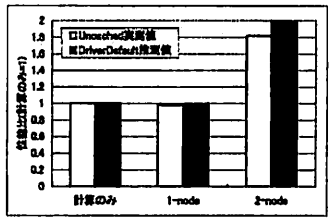

図 5 分散行列乗算結果 (N=1024)

上の時は，再送に関しての所要時間および回数である．再送にかかる時間 Tr_i, Tw_i は，再送回数が増えるにしたがい増える．最終的には，DIMMnet-2 ではソフトウェアによる再送制御は必要なくなるため，i が 1 以上の時 Nr_i, Nw_i は 0 となる．キャッシュ属性を DriverDefault にした場合の Tw_i, Tw_i は，図 3 のデータを用い，転送時間の合計 T_{tr} を推定している．T_{os} はページフォールトによる OS 呼出しと，データのフラッシュによる OS 呼出しのオーバヘッドであり，ページフォールトによる OS 呼出しは 1 回あたりのページフォールトによるオーバヘッドの計算値 T_{mmap} とページ数の計算値 N_{page} の積，データのフラッシュによる OS 呼出しのオーバヘッドは，1 回あたりのフラッシュのオーバヘッドの計算値 T_{flush} とフラッシュ回数の計算値 N_{flush} の積から算出される．これらにはディスクへの転送時間は含まれない．$T_{barrier}$ はバリア同期に要する時間の合計で計算値である．$T_{barrier}$ は実測により，最低 1[ms] かかることが計測されており，各ノードでの可変値で各ノードでの T_{all} が同一になるような数値になる．T_{calc} は，OS のオーバヘッド，ディスクへの転送時間，バリア同期に要する時間を除いた計算時間の総計の実測値である．

(1) 行列乗算

単純な分散処理による評価として，求める行列 (N*N 行列) を列方向に P 個に分割し，分散処理を行い，行列積を求める所要時間を計測した．結果を図 5 に示す．なお，ページフォールト回数 N_{page} は下記式で求めた。$size\_dbl$ は double 型のサイズを示す．

$$N_{page} = (N^2 + N^2/P) * size\_dbl/4096 \qquad (10)$$

行列乗算では計算に要する総時間 $O(n^3)$ に占めるディスクアクセスの割合が少なく，また，各ノードでの計算量が同一な上，同期を取る必要がほとんどないため，非常に効率的に並列化が行える．今回の場合においても，2 台での計算では約 1.99 倍の性能を実現できる．

(2) **Wisconsin Benchmark**

Wisconsin Benchmark はデータベースの性能を測定するベンチマークである．図 6 のようなクエリに関して，C 言語で記述し，規定された表形式のデータの格納先を DIMMnet-2 の SO-DIMM およびハードディスクに変えて性能測定を行った．Wisconsin Benchmark を分散処理させた結果を図 7，なお，ページフォールト回数 N_{page} は下記式により求めた．N_{tuple} はタプル数の合計である．

$$N_{page} = \sum_{i=0}^{n} (N_{tuple_i} * tuple\_size)/4096 \qquad (11)$$

単純な join のないクエリでは，ディスク性能が大きく影響し，Q1 では HDD に対して 2.49 倍の性能向上となっている．逆に分散処理が有用に働くのは join があるクエリ (Q4,Q6) であり，Q4 の場合には，1 ノードの場合に対して 1.97 倍の性能向上となった．データベースのような膨大なデータの格納が要求される分野においても AT-dRAM と DIMMnet-2 は有効である．

```
(Q1) select * from tenk1 where (unique2 > 301) and (unique2 < 402)
(Q3) select * from tenk1 where unique2 = 2001
(Q4) select * from tenk1 t1,tenk1 t2 where
     (t1.unique2 = t2.unique2) and (t2.unique2 < 1000)
(Q7) select MIN(unique2) from tenk1
```

図 6 Wisconsin Benchmark クエリ (抜粋)

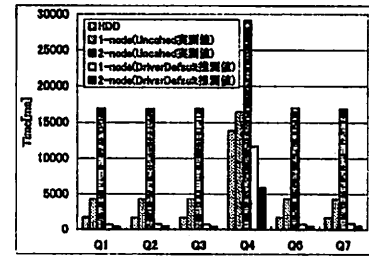

図 7 Wisconsin Benchmark 結果

7. おわりに

本研究では，Windows 向けの DIMMnet-2 用ドライバを開発した．さらにクラスタシステムを構築し，分散処理に関する評価を行った．結果，Windows 上での DIMMnet-2 による分散処理を行うことができ，評価実験からは分散処理により 2 ノードで最大 1.99 倍の性能向上が予測でき，クラスタシステムとしての有用性を示すことができた．また，本研究では DIMMnet-2 および，Windows を対象として議論を行ったが，本論文での提案は DIMMnet-2 および，Windows のみに限定されるわけではない．OS のカーネルに手を加えることができないコモディティOS 環境下で，デバイスドライバとメモリマップトファイルの組み合わせによる分散処理に特化したハードウェアでの分散処理の可能性を示したものであるといえる．

参考文献

1) W. Gropp, E. Lusk: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard, Parallel Computing, Vol. 22, No. 6, pp. 789–828 (1996.9).
2) P. Keleher, S. Dwarkadas, A. L.Cox, and W. Zwaenepoel: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, Proc. of the Winter 94 Usenix Conf., pp. 115–131 (1994.1).
3) 北村, 濱田, 宮部, 伊澤, 宮代, 田邊, 中條, 天野: DIMMnet-2 ネットワークインタフェースコントローラの設計と実装, IPSJ, Vol.46, No.SIG 12, pp.13–26 (2005.8).
4) 丹羽, 松本, 平木: コンパイラが支援するソフトウェア DSM 機構: ADSM と UDSM の性能評価, IPSJ 99-HPC-77 (SWoPP'99), Vol. 99, No. 66, pp. 95–100(1999.10)