B-049

MPI を利用した分散大容量メモリシステムにおけるページスワッププロトコルの評価
The Evaluation of Page Swapping Protocols on MPI for the Distributed Large Memory System

古尾谷 歩† 緑川 博子† 甲斐 宗徳†
Ayumu Furuoya Hiroko Midorikawa Munenori Kai

1. はじめに

分散大容量メモリ（以下 DLM）[1]とは，クラスタ内にある各ノードのメモリを集約して，一つの仮想メモリとして逐次プログラムから利用可能にするシステムである．このようなシステムが開発された背景には，64bitOS の普及によりアドレス空間が飛躍的に増大したことと，ネットワークが高速化し，従来仮想記憶として使用していたハードディスクの入出力速度を上回ったことが挙げられる．

DLM においてユーザプログラムは計算ノードという単一のノードで実行され，残りのノードはメモリサーバとして稼働する．計算ノードにないデータにユーザプログラムがアクセスする場合には SIGSEGV が発生し，SIGSEGV ハンドラによって計算ノードとメモリサーバノードとの間でデータのスワップが行われる．この間，ユーザプログラムは停止しているため，スワップ時間は DLM の性能に影響する．

そこで我々は新しいスワップの方式（スワッププロトコル）を考案し，従来のスワッププロトコルと比較した．

2. ページスワッププロトコル

DLM では DLM ページ（以下単にページ）という単位でメモリを管理しており，スワップもページ単位で行われる．図 1 に示すように，ユーザプログラムは計算ノード内の計算スレッドで実行されており，計算ノード内に確保されていないページにアクセスをした場合には，通信スレッドに依頼してメモリサーバとの間でページのスワップを行う．これには①メモリサーバへのページ要求，②メモリサーバからの必要なページの取得「スワップイン」，③メモリサーバへの不要なページの送出「スワップアウト」の 3 つの処理がある．大きく分けて，スワップ方式には ①②③の順で行うシリアルプロトコル（図 1）と①③②の順で行うクロスプロトコル（図 2）という 2 つが考えられる．

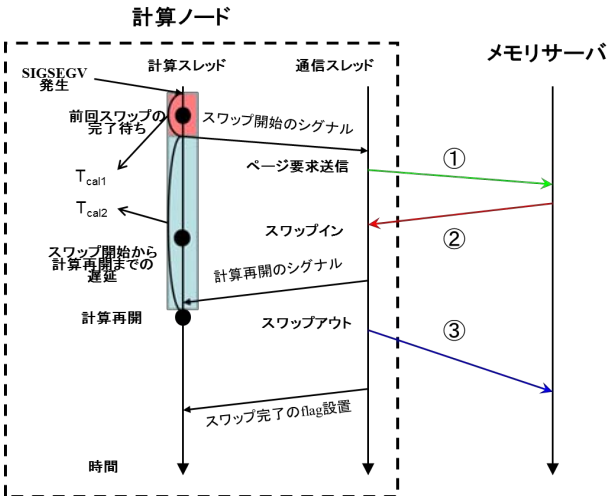


図 1 シリアルプロトコル

従来は図 1 のようなシリアルプロトコルを採用していた．まず，通信スレッドはメモリサーバへのページ要求の送信 MPI_Send()とスワップインの受信 MPI_Recv()を順に行う．スワップインが完了した時点で計算スレッドは計算を再開する．それと並行して通信スレッドでは，MPI_Send()によりスワップアウト処理行う．ただし，SIGSEGV 発生時に，もし前回のスワップアウトが完了していない場合は，前回のスワップアウトが終了するまで，新しいスワップ処理を開始せずに待つこととしている（図 1 の T_{cal1} に相当）．

ユーザプログラムから見ればスワップインさえ完了すれば計算は続けられるため，最も早くプログラム再開ができて効率が良いと考えられる．しかし，プログラムが使用するデータの大部分がメモリサーバ（遠隔メモリ）にある場合には，SIGSEGV ハンドラの発生頻度（スワップ頻度）が増えて，前回のスワップアウトが完了しておらず，その完了を待つ可能性が増えてしまう．このため，ローカルメモリ率（プログラムの使用するメモリサイズに対するローカルメモリサイズの割合）が小さい時，現状のシリアルプロトコルは効率が悪くなることがある．

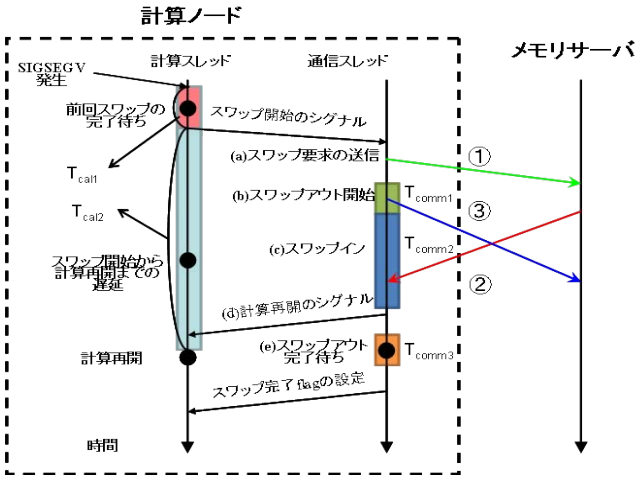


図 2 クロスプロトコル

そこで今回，全二重通信を利用するクロスプロトコルを新たに設計し，スワップの効率化を図る．通信スレッドは図 2 に示すように，最初にページ要求をメモリサーバに MPI_Send()で送信(a)し，次にノンブロッキング通信 MPI_Isend()関数(以下 Isend) でスワップアウト(b)を行う．ノンブロッキング通信を用いることで，スワップアウト完了を待たずに次のスワップイン通信を開始することができる．スワップイン(c)はブロッキング通信 MPI_Recv()（以

†成蹊大学 理工学研究科 理工学専攻 Graduate School of Science and Technology, Seikei University

下 Recv）で行い，スワップインが完了した時点で計算スレッドにユーザプログラム再開のシグナルを送る(d)．その後，MPI_Wait(以後 Wait)によりスワップアウトを完了する(e)．もし，スワップアウトが完了していない場合はこの時点で待つことになるが，通信が二重になっているためスワップインの完了を持つ可能性は少ないと考えられる．メモリサーバについても，要求ページの送信には Isend を用いた．スワップアウトされるページの受信は Recv を用いた．

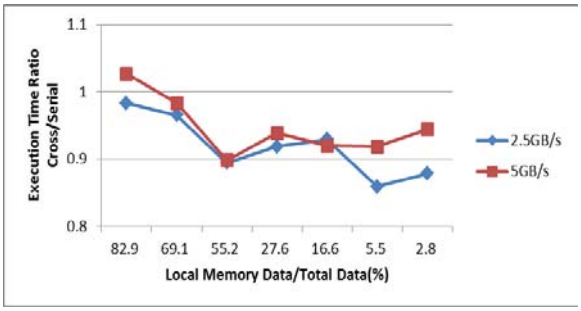
3. 評価実験

3.1 実験環境

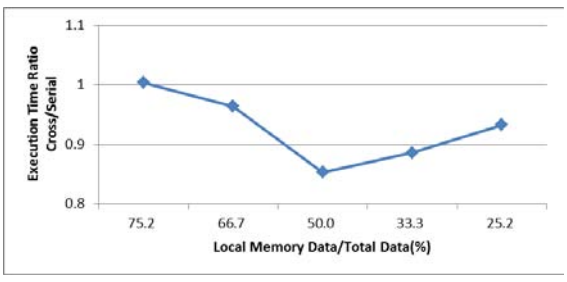
2 つのプロトコルについて，ローカルメモリ率を変えて処理時間がどのように変化するかを調査した．実験には T2K-Tokyo，HA8000 クラスタ（表 1）[3]を用いた．実験では，姫野ベンチマークと NAS Parallel Benchmarks の C プログラム版である NPB2.3-omni-C[4] の BT(NPB.BT)を用いた．姫野ベンチマークについてはネットワークバンド幅 5 GB/s と 2.5GB/s の 2 つの環境で，NPB.BT は 5 GB/s で実験を行った．問題サイズは姫野ベンチマークでは LARGE，NPB.BT では Class A で，必要なメモリサイズはそれぞれ 1810MB，306MB である．NPB.BT については 200 回の繰り返しを 10 回に短縮し，計算検証（verify）部は省略した．

表 1　HA8000 クラスタシステム

	T2K Open Supercomputer, HA8000
Machine	HITACHI HA8000-tc/ RS425
CPU	AMD QuadCore Opteron 8356(2.3GHz) 4 CPU/ node
Memory	32 GB/ node (936 nodes)，128 GB/ node (16nodes)
Cache	L2 : 2 MB/ CPU (512 KB/ Core), L3 : 2 MB/ CPU
Network	Myrinet-10G x 4, (5 GB/s full-duplex) Myrinet-10G x 2, (2.5 GB/s full-duplex)
OS	Linux kernel 2.6.18-53.1.19.el5 x86_64
Compiler	gcc version 4.1.2 20070626 mpicc for 1.2.7

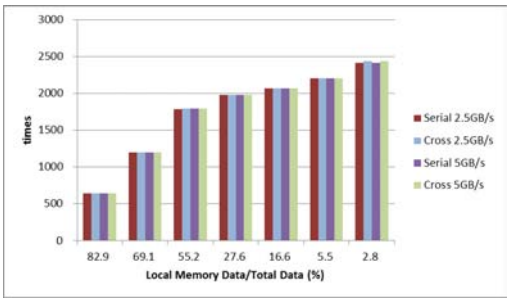


(a) 姫野ベンチマーク

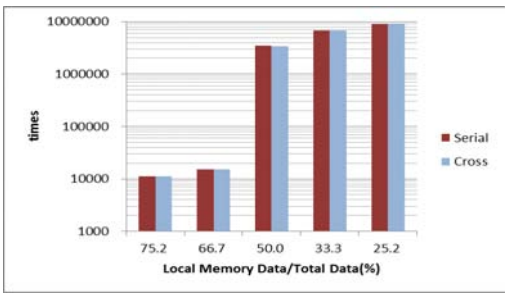


(b) NPB.BT(5GB/s)

図 3 シリアルプロトコルとクロスプロトコルの実行時間比



(a) 姫野ベンチマーク



(b) NPB.BT(5GB/s)

図 4 SIGSEGV 発生回数（スワップ回数）

3.2 スワッププロトコルの実行時間への影響

図 3 は，各プログラムにおけるシリアルプロトコルとクロスプロトコルの実行時間比である．横軸はローカルメモリ率を表し，縦軸はシリアルプロトコルでの実行時間に対するクロスプロトコルの相対実行時間を表す．姫野ベンチマークでは一回の試行に要した時間の平均実行時間とした．

ほぼすべての局面において，従来のシリアルプロトコルよりも実行時間が短縮された．特に，ローカルメモリ率が小さい場合には時間短縮の効果が大きい．相対時間が最短になったのは，(b)NPB.BT でローカルメモリ率 50.0%のときで 0.85 になった．シリアルプロトコル使用時よりも実行時間が長くなったのは，(a)姫野ベンチマークの 5GB/s，ローカルメモリ率 82.9%の場合のみで，相対実行時間は 1.03 であった．この理由は，シリアルプロトコルが最も速く動作する条件を備えていたためである．すなわち，ローカルメモリ率が高いためにスワップ頻度が少なく，通信バンド幅が高いためページの転送も早く終了し，前回のスワップアウト処理を待つ機会がほとんどない．

3.3 スワップ回数

図 4 は各プログラムの実行中の総スワップ回数を表している．横軸はローカルメモリ率である．どちらのプログラムでも，ローカルメモリ率が低下するとスワップ回数が増大する．(b)NPB.BT については，縦軸のスワップ回数は対数表示になっており，ローカルメモリ率 66.7%と 50.0%との間では 230 倍，75.2%と 25.2%との間では 800 倍にもスワップ回数が増加している．これはローカルメモリサイズが極端に小さいため BT のメモリアクセスワーキングセットがローカルメモリに収まらなくなったため考えられる．このため，図 3(b)の BT でローカルメモリ率 50.0%のとき，最もクロスプロトコルの効果が高かったのは，スワップ頻度が急激に高まったせいと考えられる．

3.4 計算スレッドにおけるスワップ処理時間

図 5，6 は，姫野ベンチマーク，図 7 は NPB.BT の場合の計算スレッドにおける SIGSEGV ハンドラの処理時間の内訳を示す．横軸はローカルメモリ率，縦軸は 1 回のスワップにおける計算スレッドのスワップ処理時間成分を表している．(a)はシリアルプロトコル，(b)はクロスプロトコルで，sigwait(previous)は前回のスワップの終了待ち時間を表し，それぞれ図 1 図 2 の T_{cal1} に対応した時間である．sigwait(swapin)はスワップインが終了するまでに要した時間であり同じく T_{cal2} に対応する．なお，（b）のクロスプロトコルは，スワップアウトを行いながらスワップインを行うため，T_{cal2} にスワップアウトの時間が含まれている．一方，（a）のシリアルプロトコルの T_{cal2} には，ユーザプログラム再開後，通信スレッドが並行して行っているスワップアウトの処理時間が含まれていない．

・前回スワップアウトの終了待ち時間

いずれの応用でも，(a)シリアルプロトコルではローカルメモリ率が低下するにつれ，前回のスワップアウト待ち時間 sigwait(previous)が増大した．ローカルメモリ率が低い状態だとスワップ頻度が増加し，その結果，前回のスワップアウトの完了を待つ時間が増えることが確認された．姫野ベンチマークの 2.5GB/s（図 5）では，ローカルメモリ率 27.6%でスワップ処理全体に占める前回待ち時間の割合は 32%である．より通信性能が高い 5GB/s（図 6）では少し緩和され，ローカルメモリ率 2.8%のとき全体の 25%を前回待ち時間が占める．NPB.BT（図 7）では，非常に多くのスワップが発生しており，ローカルメモリ率 33.3%のとき，スワップ処理全体に占める前回待ち時間の割合はスワップ処理全体の 45%にも達する．

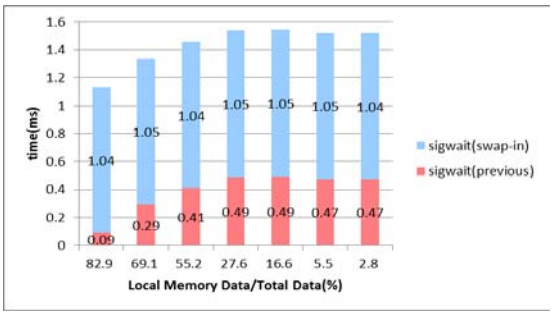
(b)クロスプロトコルでは前回の待ち時間は小さくなった．通信を二重にしてスワップアウトを前倒ししたことで，前回のスワップアウトを待つ状況を低減している．特に，姫野ベンチマーク（図 5,6）では前回待ち時間が 0 で，前回のスワップアウトを待つ状況は起こらなかったといえる．NPB.BT（図 7）でも前回待ち時間はローカルメモリ率 50.0%より小さい時のみ 0.11ms（全体の 8%）だった．

・スワップインに要する時間

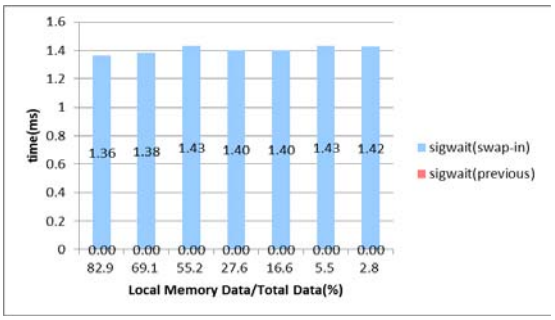
スワップイン時間 sigwait(swapin)を，図 5~7 のそれぞれで(a)と(b)とを比べると，1.3～1.5 倍クロスプロトコルの方が大きくなった．全二重通信により，スワップイン受信と並行してスワップアウト送信が行われるため，スワップアウトの起動時のオーバヘッドにより，シリアルプロトコルの場合のスワップイン単一の通信に比べ，クロスプロトコルではスワップイン完了までの時間が増加している．

スワップイン時間はローカルメモリ率の変化にはあまり影響されなかった．ただし，NPB.BT のクロスプロトコルでは（図 7(b)），ローカルメモリ率が低下するにつれ，スワップイン時間が 5%程度増加している．

姫野ベンチマークの 2.5GB/s（図 5）では，(a)シリアルプロトコル(1.04ms)と比べ，(b)クロスプロトコルはスワップイン時間(1.42ms)が 1.36 倍に増加した．同様に 5GB/s（図 6）ではシリアルプロトコル(0.80ms)がクロスプロトコル(1.14ms)へと 1.43 倍にスワップイン時間が増えた．NPB.BT（図 7）では，ローカルメモリ率 66.7%ではシリアルプロトコル(0.80ms)からクロスプロトコル(1.17ms)へ

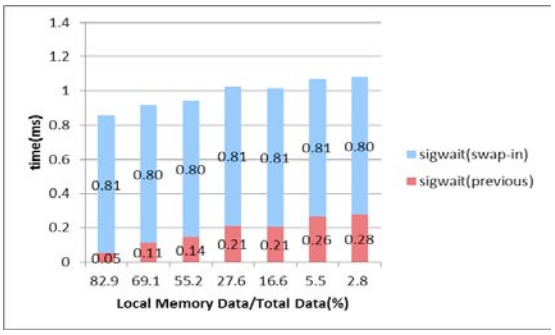


(a)　シリアルプロトコル

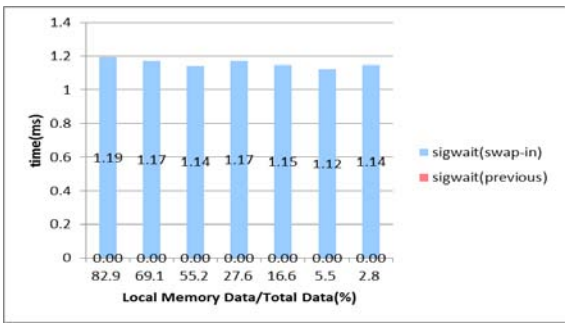


(b)　クロスプロトコル

図 5　姫野ベンチマークにおける計算スレッドのスワップ処理時間成分(2.5GB/s)



(a)　シリアルプロトコル



(b)　クロスプロトコル

図 6　姫野ベンチマークにおける計算スレッドのスワップ処理時間成分(5GB/s)

1.46 倍に増加し，ローカルメモリ率 25.2%ではシリアルプロトコル（0.81ms）が クロスプロトコル(1.23ms)へとスワップイン時間は 1.52 倍になっている．

3.5　通信スレッドにおけるスワップ処理時間

クロスプロトコルについて，通信スレッドにおけるスワップ処理時間の内訳を分析した．図 8 はその結果を表す．

縦軸は，1 回のスワップにおける通信スレッドの処理時間成分を表している．図中の start swapout とは，スワップアウトを開始するノンブロッキング通信の呼び出しに要した時間であり，図 2 の T_{comm1} に相当する．図の swapin はスワップインに要した時間であり，同じく T_{comm2} に相当する．wait swapout はスワップインの完了後，スワップアウトの終了を待つ処理の所要時間であり同じく T_{comm3} に相当する．この時間は計算スレッドの T_{cal2} に含まれない．T_{comm3} の後に行われるスワップアウト済みのページの unmap にも時間を要したので図に加えている．(a)姫野ベンチマークでは，2.5GB/s と 5GB/s でそれぞれ代表としてローカルメモリ率 82.9%の場合と 5.5%の場合を表示し，(b)NPB.BT では計測したローカルメモリ率を全表示した．

wait swapout はいずれのプログラムでも 0.01ms と非常に短かった．このことから，スワップインが完了した時点ですでにスワップアウトも完了していると考えられる．その一方で，start swapout は最小でも 0.18ms を要し，スワップアウトを開始するために呼び出す Isend の処理には時間がかかっていることが分かった．

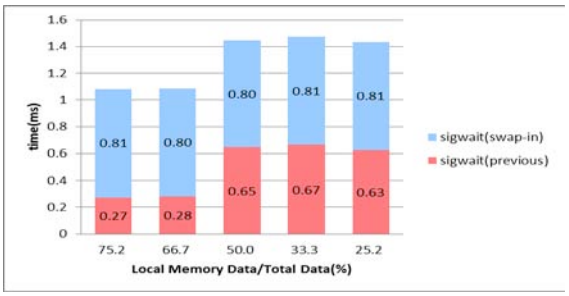
このことから，Isend の処理時間の分だけ，スワップアウトの通信がスワップインの通信よりも早く始まり，通信の完了もスワップアウトの方が早く終わると考えられる．そこで，現クロスプロトコルのスワップインのブロッキング受信をノンブロッキング受信へ変更し，(1)スワップインページ Irecv ，(2)スワップアウトページ Isend，(3)スワップインに対する Wait（終了時にユーザプログラム再開），(4)スワップアウトに対する Wait の順で通信を行うプロトコルも考案した．しかし，DLM を使わず 2 つのノード間で 1MB データを繰り返し交換する MPI テストプログラムで予備実験をしたところ，スワップを開始してからスワップインが完了するまでの時間は現プロトコルと同程度であることがわかったため，今回の評価では省いた．

NPB.BT をクロスプロトコル，ローカルメモリ率 50.0%以下で実行したときの前回待ち時間は，図 7(b)より 0.11ms であった，一方，同じ条件で通信スレッドがスワップイン完了後に行う処理の所要時間は，図 8(b)の wait swapout と unmap の合計より 0.11ms で，ほぼ一致している．

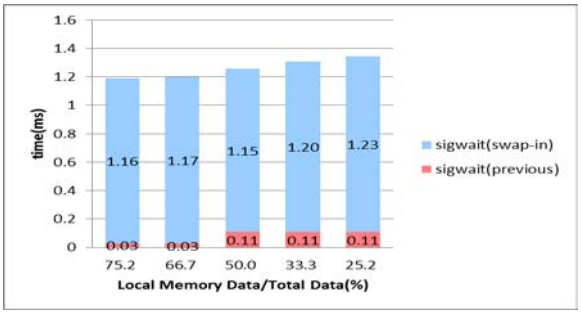
4. まとめ

クロスプロトコルの採用によって前回のスワップアウトの待ち時間を大幅に短縮することができた．プログラム全体の実行時間も，クロスプロトコルの採用によってシリアルプロトコル使用時と比べて最短で 85%にまで短縮された．ローカルメモリ率が高くスワップがあまり発生しない状況では，シリアルプロトコルも高速であるが，通信バンド幅が低い環境や，ローカルメモリ率が低くスワップが頻繁におこる状況では，クロスプロトコルはシリアルプロトコルに対しての優位性がある．

今後，このプロトコルにページのプリフェッチ機能などを加え，さらに高性能化を目指す予定である．

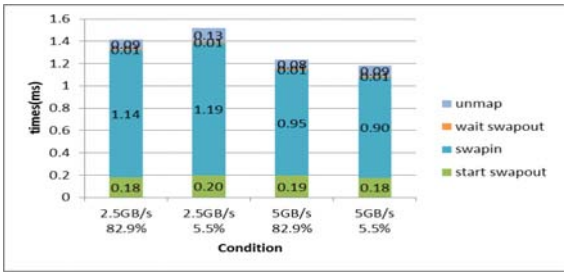


(a)　シリアルプロトコル

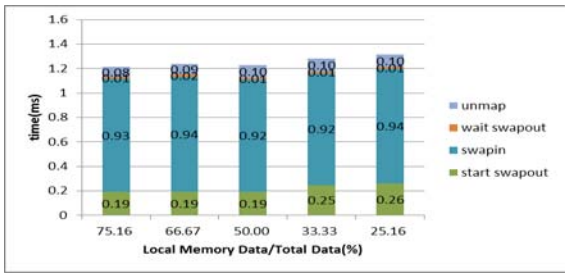


(b)　クロスプロトコル

図 7　NPB.BT における計算スレッドのスワップ処理時間成分(5GB/s)



(a) 姫野ベンチマーク



(b) NPB.BT(5GB/s)

図 8　クロスプロトコルにおける通信スレッドのスワップ処理時間成分

参考文献

[1] 緑川博子，齋藤和広，佐藤三久，朴 泰祐："クラスタをメモリ資源として利用するための MPI による高速大容量メモリ"，情報処理学会論文誌，コンピューティングシステム，Vol.2, No.4, pp.15-36, (Dec.2009)
[2] 姫野ベンチマーク http://accc.riken.jp/HPC/HimenoBMT.html
[3] NAS Parallel Benchmarks NPB2.3-omni-C http:/www.hpcs.cs.tsukuba.ac.jp/omni-openmp/
[4] T2K-Tokyo, HA8000 クラスタシステム. http://www.cc.u-tokyo.ac.jp/ha8000/