マルチクラスタ向けソフトウェア分散共有メモリの提案

吉川 克哉[†]　城田 祐介[†]　吉瀬 謙二[†]　本多 弘樹[†]　弓場 敏嗣[†]

複数のクラスタシステムを繋げたマルチクラスタが並列処理の共通アーキテクチャになっていくことが今後期待されている。しかしマルチクラスタでは、クラスタ間通信遅延がボトルネックとなってしまう。本稿では、単一の PC クラスタ上でマルチクラスタをエミュレートするシステムを構築し、同システム上で既存のソフトウェア分散共有メモリのライトバック型のページ一貫性制御方式をそのまま使用した場合のオーバヘッドを明らかにする。クラスタごとにホームノードを設け、同ノードをクラスタキャッシュとして利用することでクラスタのデータローカリティを利用し、同オーバヘッドの隠蔽を図るマルチホーム方式を提案･実装し、その有効性について検証する。この結果、既存のソフトウェア分散共有メモリ方式に対し最大 38.5%の速度向上が得られた。さらに、マルチホーム方式を用いて大規模クラスタシステムを論理的なマルチクラスタとみなし、共通のアプローチを適用すると、ホームノードからのページ読み出しが複数ホームに分散され、同読み出しのコンテンションが緩和される効果が得られ、低速なネットワーク環境においても従来方式では実現できなかった高いスケーラビリティを得ることができた。

Proposal of Software Distributed Shared Memory for Multiclusters

Katsuya Yoshikawa,[†] Yusuke Shirota,[†] Kenji Kise,[†] Hiroki Honda[†] and Toshitsugu Yuba[†]

As multiclusters, or aggregate PCs clusters, are increasingly becoming an appealing platform for parallel computing, software distributed shared memory(SDSM) layer across them is quite an attractive alternative for parallel application programmers. This paper's goal is to provide the potential for application performance superior to that achievable on a single cluster system. To achieve such performance on multiclusters for parallel applications, adding new functions to conventional home-based SDSMs is required for reducing inter-cluster traffic and hiding inter-cluster latency. This paper introduces multi-home scheme, a method to set per-cluster home nodes to exploit cluster data locality, and integrates this extension to JIAJIA, a home-based SDSM system suitable for multiclusters. Evaluations are done on our pseudo multicluster built on a single PCs cluster, and evaluation result shows multi-home scheme achieves high performance, up to 38.5 % in comparison to conventional single home schemes. Applying this concept to virtual sub-clusters of a large cluster system which consists of thousands of processors, allows page requests to be distributed to multiple home nodes, resulting in reduction of shared page access contentions induced on home nodes, and realized high scalability on a 100base-TX Ethernet based PCs cluster in spite of its high communication overheads.

1. はじめに

プログラムを並列化し並列処理する上で、スーパークラスタ[1]やグリッドを利用し複数のクラスタシステムを繋げたマルチクラスタ（MC）が共通アーキテクチャになっていくことが今後期待されている。このようなシステムを効率良く利用するためには並列プログラミング環境の構築が重要である。分散メモリシステム上に仮想的な共有メモリの構築をユーザレベルのソフトウェアライブラリで実現するページベースソフトウェア分散共有メモリ(SDSM)は、分散メモリシステム上で共有メモリプログラミングモデルを提供する一方で、パフォーマンスのスケーラビリティが得られないため実用的なレベルに達しているとは言い難い。しかし、SDSM の用途はアプリケーションプログラマが直接 SDSM コードを記述するだけのものではない。例えば、従来は共有メモリシステムで実現されていた OpenMP が、SDSM に対して OpenMP プログラムをコンパイルすることでクラスタ上で実現されている[2]ように、スケーラブルな SDSM の実現が望まれている。

そこで、我々はまず、MCを PC クラスタ上でエミュレートするシステムを構築し、予備評価として既存のページベース SDSM のライトバック型のページ一貫性制御方式を同システム上でそのまま使用した場合のオーバヘッドを求める。MC上でページベース SDSM を利用する場合、ページ転送などでクラスタ間のトラフィックが頻繁であるため、本稿ではキャンパスグリッド規模までのMCを前提とする。そして、明らかになったオーバヘッドを削減する足掛かりに、ライトバックの戻り先であるホームノードをクラスタごとに複数設け、個々のクラスタのデータローカリティを利用することでクラスタ間通信遅延の隠蔽を図ったページ一貫性制御方式としてマルチホーム（MH）方式を提案し、既存の SDSM 上に実装して、予備評価をおこなう。

SDSM が広く利用されるためには、メッセージパッシング方式などに対して、SDSM がコードの記述性だけでなく、期待し得る最大のスケーラビリティや速度性能を示し、実用面での優位性を提示することは重要である。本稿では、MH方式のページ一貫性制約の緩め方をページごとに適宜変化させることで、同方式が合わせ持つ複数ホームノード間の一貫性をとるオーバヘッドを除去した最適な場合にさらに性能とスケーラビリティに大幅な向上が得られることを示す。MH方式を用いた場合に期待しうる最大性能として、SDSM プログラムを仮実行して得られるメモリ参照履歴情報を利用して、SDSM のソフトウェアライブラリレベルでの最適化をおこなった結果を示す。SDSM のソフトウェアライブラリレベルでは、データ通信などがページ単位で管理されるため、比較的容易にページごとに最適になるようにソフトウェアライブラリの再構築をおこなうことができる。

また、クラスタ性能によってMH方式の性能を高める実装方法は異なる。多彩なクラスタ性能に対応するために、クラスタごとに設けるホームノードの最適な数を算出するためのテストプログラムを実行し、同数を決定することで、クラスタ性能に固有なソフトウェアライブラリを構築する機構を提案し、予備的評価をおこなう。

また、MH方式を用いることによって、大規模クラスタシステムを、クラスタ間通信遅延がスイッチングディレイのみの論理的なMCとしてみることができる。大規模クラスタシステムでのスケーラブルな SDSM の実現に向けて、MH方式が大規模な並列システムとして標準的な SMP クラスタシステムでもスイッチレベルでのコンテンションを緩和する場合があることを予備評価で示す。

さらに、バリア同期などの一貫性制御機構の制御用データ転送が、既存の SDSM の一貫性制御方式をMC環境

† 電気通信大学　大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

で用いた場合に無駄な同期待ち時間やコンテンションを引き起こすことを解析した結果示す。この解析結果をもとに、SDSM の制御用データ転送について一種のスケジューリングをおこない、MH方式と組み合わせることで、飛躍的な速度向上が得られることを示す。共通のアプローチで大規模クラスタシステムにも適用できると考える。

以下, 本稿では, 2章では、MC上で SDSM を実行する問題点を述べる。3章ではMH方式、4章では同方式の実装方式などを説明する。また, 5章では予備評価用のMCエミュレータの実装方法、及び、ベンチマークプログラムによる性能予備評価結果を示す。6章で関連研究を概観し、7章で今後の課題を述べる。

2. マルチクラスタで既存の SDSM の一貫性制御方式を用いる問題点

2.1 SDSM の選択

MCとの親和性などを考慮した上で、既存の各種 SDSM から JIAJIA[3]を選択する。JIAJIA はページ一貫性制御方式に、ホームノードがページごとにある固定ノードに決められているホームベース方式を採用したため、総合的に高性能なばかりでなく、使用可能なメモリサイズが 1 ノードの物理メモリサイズに限定されないので、大規模問題を前提とする我々の要求に合致する。また、我々の提案している権限委譲プロトコル[6]を用いれば、ホームベース方式のライトバックによるオーバヘッドが最も顕著化する複数ノード間で頻繁に排他的にページを更新・読み出すメモリ参照パターンである Migratory access パターンにおいて同オーバヘッドを削減できる。また、メモリの読み出しに際し、TreadMarks[4]の diff 分散方式がページ差分情報である diff 単位でおこなうのに対して、ホームベース方式ではページ単位でおこなわれるため履歴情報として扱い易く、各種プロファイリング技法を比較的容易に適用できるなど多くの利点を有する。

2.2 マルチクラスタ上での実行に伴うオーバヘッド

JIAJIA や TreadMarks の実装は、他の多くの各種 SDSM と同様にすべての CPU をフラットな構成とみるものになっている。このため、TreadMarks の一貫性制御方式をMC上でそのまま用いると頻繁なページ転送などでクラスタ間通信遅延が性能ボトルネックになってしまう[5]。JIAJIA についても同様の結果になることが予想される。

3. マルチホーム方式の提案

3.1 クラスタキャッシュ

MCにおけるクラスタ間通信遅延を隠蔽するために、クラスタのデータローカリティを利用する。具体的には、更新されたページをクラスタごとにクラスタ内のあるノードにキャッシュし、同キャッシュをクラスタキャッシュとして利用する。これにより、本来ならばクラスタ外のホームノードとのページ授受が必要なページフォルト処理をクラスタ内で解決することが可能になる（図1）。

また、大規模クラスタシステムを論理的なMCとみなし、同様のアプローチを適用すると、ページ読み出しが複数クラスタキャッシュに分散され、同読み出しのコンテンションが緩和される効果が得られることが期待される。

3.2 クラスタごとのホームノード

本稿では、「クラスタごとのホームノード」という概念を導入し、同ノードをクラスタキャッシュとして利用することを検討する。このクラスタキャッシング方式を、クラスタごとのホームノードに対してライトバックをおこなうMH方式とそうでないプロキシホーム方式に分類する。これに対して従来方式をシングルホーム（SH）方式と呼ぶ。厳密なMH方式では、ライトバックを積極的にすべてのクラスタのホームノードに対しておこなうので、複数のホームノード間の一貫性をとるオーバヘッドが大きいことは明らかである。このため、必ずしもパフォーマンス向上は見込めないが、クラスタキャッシュとしての有効性の検証という目的で我々はまずMH方式を実装する。また、本稿では、MH方式のページ一貫性制約を適切に緩めた場合に得ることができる性能も提示する。

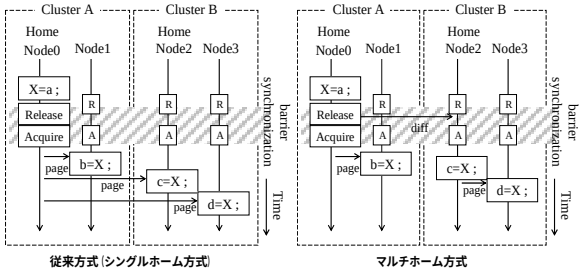

図 1：シングルホーム方式とマルチホーム方式のバリア同期機構
Fig1：Barrier synchronization of single-home scheme and multi-home scheme

その後、より効率的なクラスタキャッシング方式であるプロキシホーム方式について検討する。プロキシホーム方式でのキャッシングはライトバック時におこなうのではなく、クラスタ外のホームノードから一度受け取ったページをキャッシュし、再利用することなどが考えられるが、プロキシホーム方式の詳細については本稿では議論しない。

4. マルチホーム方式の実装

4.1 マルチホーム方式の実装方法

本稿ではまず、積極的にすべてのホームノードに対してライトバックをおこなう厳密な実装方式を採用する。クラスタごとのホームノードの数については最適値があり得るが、まずはホームノードの数を各ページについてクラスタごとに1つとし、すべてのクラスタの構成がホモジニアスであるとの前提のもと、ホームノードをクラスタ間で対称となるように配置する。ページリクエストの発行は個々のクラスタ内のホームノードに対しておこなう。

4.2 実装の概要

JIAJIA にMH方式を実装する。図 1 で JIAJIA における従来のSH方式と、MH方式のバリア同期機構の実装方法を比較する。SH方式では、バリア同期操作を発行する際、その直前のバリア同期区間において、各ノードがあるページに書き込みをおこなった場合、書き込みをおこなう前とおこなった後のページの差分（diff）でホームノードにライトバックし、ライトバック完了確認を待つ。同完了確認後、バリア同期機構の集中スケジューラノードにバリア同期到達通知と書き込みをおこなったページについての書き込み通知をまとめて送信する。JIAJIA のSH方式の実装に対して、MH方式では、すべての複数ホームノードに対して diff でライトバックするようにした。また、JIAJIA のSH方式では、ページのホームノードは書き込みに際し、diff の作成・送信の必要がなかった。しかし、MH方式では、ホームノードが複数あるため、ホームノードが書き込む場合でも他のホームノードに対して diff を送信する必要がある。

ロックを用いた排他制御機構の実装方法についても同様に、ライトバックを複数ホームノードに対しておこなう。

4.3 クラスタ性能に固有な実装方法を実現する機構の提案

一般的に、ノード数やネットワーク性能などの構成要素が規定するクラスタ性能によってMH方式の性能を高める実装方法は異なると考えられる。例えば、4.1 では、あるページについて各クラスタに1つずつホームノードを設けたが、クラスタ性能に適応したホームノードの数があり得る。本稿ではこの例について、多彩なクラスタ性能に対応するための機構を提案する。

提案機構では、まず、最適なホームノード数を算出するためのテストプログラムをユーザが実行し、同値を同定する。そして、この値をパラメータとしてソフトウェアライブラリが取得し、ソフトウェアライブラリを再構築する。以上により、クラスタ性能に固有なソフトウェアライブラリを構築することが可能となる。

この際、用いるテストプログラムが問題となる。テストプログラムのパラメータとして、SDSM のページサイズ、ページ数、データとホームノードのマッピング、アプリケーションプログラムのメモリ参照パターンなどを検討し、高精度を実現させる必要がある。あるいは、メモリ参照パターンなどの特性をアプリケーションプログラムから自動的に抽出することも必要となりうる。本稿では、MH方式が Producer-Consumers 型のメモリ参照パターンで最も有効であると考えられることに着目し、同型のメモリ参照パターンを持つ簡単なテストプログラムを実験的に作成した。テストプログラムの詳細は5章で示す。

4.4 メモリ参照履歴を用いたマルチホーム方式の性能最適化

MH方式を用いた場合に期待しうる最大のスケーラビリティや速度性能を「MH方式の最適化」により求める。

厳密なMH方式では、ライトバックを積極的にすべてのクラスタごとのホームノードに対しておこなうので、複数ノード間の一貫性をとるオーバヘッドは大きい。

しかし、ページごとに適宜にこの一貫性制約を緩めることで、同オーバヘッドを除去することができる。換言すれば、個々のページについて、アプリケーションプログラムの実行において、基本的にはSH方式で処理を進め、MH方式が有効な部分についてのみSH方式からMH方式に切り替える。例えば、ホームノードである Producer のページへの書き込みの完了をその他の Consumers がバリア同期成立まで待ち、バリア同期成立後 Producer からこれを読み出すメモリ参照パターンにおいては、バリア同期時に対象ページに関して複数ホームノードへライトバックしSH方式からMH方式に切り替え、コンテンションを緩和したり、クラスタ間通信遅延を隠蔽することが効果的であると考えられる。このように、MH方式への切り替えが効果的となるメモリ参照パターンを検出する必要がある。

この実現方法として、アプリケーションプログラマがシステムが提供する API 等でMH方式への切り替えを明示的に指示したり、あるいは、各クラスタ内のホームノードでクラスタ外へのページリクエストを集中管理するか、SH方式の従来のホームノード側でページ読み出しのコンテンションにより検出するなど動的に検出することなども考えられるが、本稿では切り替えを最適におこなった場合に得ることができる最大性能を示すために、プロファイリング技法を用いる。プロファイリング技法にも、実行時におこなう方法と仮実行して得られるメモリ参照履歴情報を利用する方法があるが、ここでは上記理由により後者を採用する。

Producer-Consumers 等の複数ノードで一斉にページを読み出し、同読み出しのコンテンションを引き起こす可能性のあるメモリ参照パターンを検出するために取得する履歴情報とその取得方法を説明する。取得する履歴情報は、バリア同期区間ごとの読み出されたページ番号と読み出しをおこなったノード番号だけである。これで、同一のバリア同期期間でどの複数ノードが読み出しをおこなったかが分かる。必要な履歴情報を取得するためのコードを JIAJIA に挿入し、これを得る。

取得したページ番号と当該バリア同期区間をMH方式でライトバックをおこなう対象とし、これ以外のページについてはライトバックをおこなわないよう JIAJIA にコードを挿入し、ライブラリを再構築する。再構築した JIAJIA を用いてアプリケーションプログラムを再実行して得られる実行結果をMH方式の「最適」のものとする。

4.5 マルチクラスタ向けバリア同期の実装

既存の SDSM の実装では、バリア同期などの一貫性制御機構の制御用データ転送が、MC上では無駄な同期成立待ち時間やコンテンションを引き起こすことがある。よって、MH方式に限らず、バリア同期機構などの実装について、MC向けの実装が考えられる。本稿では、SDSM の性能を大きく左右するバリア同期についてMC向けの実装をおこなう。

まず、JIAJIA のバリア同期機構の実装について図 2 のMC構成を想定し説明する。バリア同期機構集中スケジューラノード（計算ノードも兼用する）を計算ノード 0 とする。JIAJIA の実装では、すべてのノードからバリア同期到達通知を受信しバリア同期成立確認すると、実行再開始通知を計算ノードの0から15までクラスタ内ノード優先で転送してしまう。MC上では、クラスタ間通信遅延があるため、スケジューラノードと別のクラスタBに属する計算ノードの 8 から 15 の実行再開始は遅れる。これらノードから再びスケジューラに転送されるバリア同期到達通知もクラスタ間通信遅延の影響を受けるので、スケジューラにさらにクラスタAからの同通知より遅れて到着する。また、例えばバリア同期区間の実行時間が短いような場合、スケジューラが実行再開始通知をクラスタBの計算ノードに転送している最中に、クラスタAの計算ノードからバリア同期到達通知あるいはページリクエストなどがスケジューラに到着し、実行再開始の転送が割り込まれてしまうコンテンションを引き起こすことを確認した。

これに対して、分散管理する方法やバリア同期区間ごとにスケジューラノードの位置をずらす方法なども考えられるが、本稿ではスケジューラが実行再開始通知をクラスタBから転送することで得られる性能向上をみる。

さらに、各ノードの計算量に応じて実行再開始通知の転送順序を変更する場合の性能向上もみる。例題として、Producer-Consumers メモリ参照パターンの例題プログラムと LU 分解プログラムについて、スケジューラが、まず Producer に実行再開始通知を転送し、その後クラスタ外のノードを優先的に転送する。これは、Producer を最初に実行再開始させ、Producer がバリア到達時におこなう複数ホームノードへのライトバックを早める。そして、同ライトバックと他ノードの計算を可能な限りオーバラップさせ、複数ホームノードの一貫性をとるオーバヘッドを削減することを図るものである(producer 優先)。実験で得られた性能解析結果をもとに、SDSM の制御用データ転送について一種のスケジューリングをおこなうことの今後の指針としたい。

5. 予備評価

5.1 予備評価用擬似マルチクラスタシステムの実装

MC を単一のクラスタシステム（以後、シングルクラスタ、SC）上でエミュレートするシステムで予備評価をおこなう。表 1 をクラスタ構成要素ノードとする SMP-PC クラスタを論

表 1：評価実験環境
Table 1：Experimental platform

CPU	PentiumIII 866MHz(x2)
Memory	1GB
NIC	100BASE-TX
OS	Red Hat Linux 7.1
Compiler	gcc 2.96

表 2：ページフォルト処理時間コスト
Table 2：Page fault handling cost

ページの取得先	コスト[μsec]
SMPノード内 (クラスタ内)	219
SMPノード間 (クラスタ内)	725
クラスタ外	3069

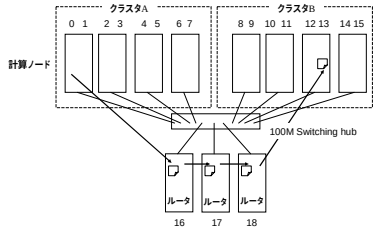

図 2：擬似マルチクラスタシステム
Fig2：Pseudo multicluster system

理的に2つの SMP-PC クラスタ構成とみなす擬似MCシステム(図 2)を実装した。

この擬似MCシステムは、クラスタ外ノードとの通信をクラスタ間ルータにみたてた複数台の SMP-PC ノードを経由しておこなわせることで、クラスタ間通信遅延をエミュレート[5]した。図 2 において、計算ノード 0 がクラスタ外の計算ノード 8～15 にページを送信する場合、ページをまずルータノード 16 に送信する。ルータノード 16 は、送信されてきたページをルータノード 17 にそのまま転送し、ルータノード 17 はルータノード 18 に転送する。ルータノード 18 は、ページの実際の送信先の計算ノード 8～15 にページを送信する。

本稿でのMCについての評価は、エミュレートするクラスタ間通信遅延が適当になるように、クラスタ間ルータノードを3つとした。エミュレートされるクラスタ間通信遅延を示すために、ページフォルト処理時間コストを測定した結果を表2に示す。ここで、ページフォルト処理時間コストは、ページアクセスによるページフォルトによりホームノードからページをフェッチし、ページアクセスが再開するまでを測定した時間である。

5.2 予備評価用ベンチマークプログラム

予備評価用ベンチマークプログラムには、行列積を求めるプログラム(MM), LU 分解プログラム(LU), NPB の IS の特性の異なる 3 つを用いた。問題サイズは表3の通りである。データとホームノードのマッピングは、SH方式の場合においてメモリ書き込みローカリティが最も高くになるようにアプリケーションプログラム側で人手で設定した。予備評価に用いたページサイズは 4K バイトとした。

5.3 予備評価結果

アプリケーションごとに、従来のSH方式とMH方式と 4.4 で定義したMH方式の最適の場合について、SC と MC 上で性能評価した結果を表4、表6、表8に示す。

5.3.1 MM の評価

表4より MM ではSC及びMC上において、従来方式では 16CPU までのスケーラビリティが得られていない。一方、

表 3：問題サイズ
Table3：Problem size

評価プログラム	問題サイズ
MM	2048x2048
IS	鍵の数=2^{26}, 鍵の最大値=2^{12}
LU	1024x1024

表 4：MM の実行時間[sec]
Table4：Execution time of MM(2048x2048)

	SC			MC		
ノード数	SH	MH	最適MH	SH	MH	最適MH
1CPU	155	-	-	-	-	-
2CPU(1+1)	89.7	89.1	82.2	110	103	90.4
4CPU(2+2)	61.3	60.1	56.0	80.5	73.1	64.4
8CPU(4+4)	40.5	41.3	38.8	60.2	54.1	47.2
16CPU(8+8)	46.2	32.9	30.8	63.8	45.6	39.3

表 5：ホームノードを4つ用いた MM の実行時間[sec]
Table5：Execution time of MM with 4 home nodes

ホームノード数 / ノード数	SC		MC	
	2	4	2	4
8CPU	41.3	42.7	54.1	69.5
16CPU	32.9	37.6	45.6	64.2

表 6：IS の実行時間[sec] (共有メモリ:4ページ)
Table6：Execution time of IS (shared memory: 4 pages)

	SC			MC		
ノード数	SH	MH	最適MH	SH	MH	最適MH
1CPU	38.2	-	-	-	-	-
2CPU(1+1)	20.0	19.8	19.7	20.3	19.9	19.9
4CPU(2+2)	10.2	10.3	10.2	12.6	12.5	12.6
8CPU(4+4)	5.64	5.65	5.57	7.36	7.17	7.26
16CPU(8+8)	3.93	3.99	3.77	5.81	5.47	5.63

厳密なMH方式では、不要なライトバックをおこなっているのにも関わらず、16CPU でも速度向上を示している。

SC上でのMH方式を用いた場合の速度向上の理由として、ホームノードでのページ読み出しのコンテンションが緩和されたためと考えられる。MC上でのMH方式を用いた場合は、これに加えて、クラスタ間通信遅延を隠蔽できた結果、高い性能向上を示している。

上記以上に注目すべきは、SH方式に対してMH方式で新たに必要となるライトバックが、SH方式のライトバックと一括しておこなわれることで、隠蔽されている点である。よって、厳密なMH方式で不必要なライトバックが増加しても、性能向上が得られるわけである。

そして、厳密なMH方式のオーバヘッドとなるライトバックを除去した最適な場合では、MCの16CPUにおいて最大 38.5%の速度向上が得られた。

また 4.3 で述べたように、ホームノードの数をクラスタごとに2つずつの合計4つにした場合の性能評価をおこなった。評価結果を表5に示す。

ホームノードをクラスタごとに2つにした場合、逆にパフォーマンスが低下してしまった。この場合、MH方式の最適な場合でも、同様の結果となった。これは、増加したホームノードへのライトバックのオーバヘッドが過大になったためである。

5.3.2 ISの評価

表6よりISでは、MC上でMH方式を用いた場合、速度向上を示した。

MC上での速度向上は、MMの場合と同様に、MH方式にすることでクラスタ間通信遅延を隠蔽できた結果である。またISでは、すべての計算ノードが同一のロック変数を使用する Migratory access パターン と、すべての計算

表 7：各アクセスパターンの実行時間[sec]
Table7：Execution time of each access pattern

8CPU(4+4)		SC		MC	
		SH	MH	SH	MH
Migratory	1ページ	0.151	0.195	0.382	0.400
	32ページ	3.13	3.96	7.93	7.44
Produder Consumer	1ページ	0.416	0.386	0.446	0.396
	32ページ	1.88	1.06	2.33	1.10

ノードがページの読み出しを一斉におこなう Producer-Consumers パターンがある。実行時間についてそれぞれのメモリアクセスパターンの内訳と Migratory access パターンにおいて読み出し･書き込みがおこなわれるページの数との関係を表7に示す。

評価結果より、SC及びMC上において、Producer-Consumers パターンではMH方式にすることで速度向上した。注目すべきは、IS では Producer が書き込みをおこなうデータがホームノードとして Producer にマッピングされているので、MM のようにMH方式が引き起こす新たなライトバックが従来方式のライトバックと一括でおこなわれることがないにも関わらず速度向上している点である。IS では、Producer-Consumers パターンのバリア同期区間において、Producer のデータ生成の実行時間が短い。よって、同区間中に計算をしない Consumers がバリア同期機構の制御用通信をおこなっている間に、Producer はデータ生成ともうひとつのホームノードへライトバックをおこなう。このオーバラップにより、同ライトバックが隠蔽され、速度向上すると考えられる。

一方、Migratory access パターンについては、SC上ではSH方式を用いた場合の性能が高いが、MC上ではページ数が増えるとMH方式の方が性能が向上することが明らかになった。

Migratory access パターンにおいて、ロックとアンロックの中で例えば表7のように32ページに対して読み出しと書き込みをおこなう場合では、SH方式では各ノードにおいて 32 回のページの無効化とページの更新が必要となる。従って、SH方式では、ホームノードがクラスタ外にある場合、ホームノードを更新するページリクエストとこれに応答するページ転送が 32 回ずつとライトバックの diff を加えたクラスタ間通信路を介した通信が必要となる。

一方、MH方式ではクラスタごとにホームノードが存在するため、クラスタ間通信路を介したページリクエストはなくなる。一方で、複数ホームノード間で一貫性をとるために、クラスタ間通信路を介した diff を用いたライトバックが多少増加する。しかし、diff は、一括して複数ページ分を送信することができるため、ページ数が増えるとそのオーバヘッドは相対的に小さくなる。

従ってISでは、SC上において、Migratory access にはSH方式、Producer-Consumers にはMH方式を採用した場合最適となる。また、MC上では、Producer-Consumers にはMH方式、Migratory access ではページ数に応じて適切な方式を決定することが最適となる。

5.3.3 LUの評価

LU では、16CPU で充分な性能が得られなかったため、8CPUで評価をおこなった。表8より、SC及びMC上で、大小の差はあれ性能低下が生じている。MH方式の最適な場合でも性能低下が生じている。

LU で厳密なMH方式を用いた場合、性能が大幅に低下している。LU では、一部のページが、バリア同期区間ごとにProducer-Consumersパターンの対象となるが、この部分も含めて、ホームノード以外のノードがページに対して書き込みをおこなわない。このような場合、SH方式が最も効果を発揮する。このため、上記一部以外のページへの大量の書き込みによるバリア同期ごとの複数ノードへのライトバックがオーバヘッドとなり性能が著しく低下する。

表 8：LU の実行時間[sec]
Table8：Execution time of LU

ノード数	SC			MC		
	SH	MH	最適MH	SH	SH (producer優先)	最適MH
1CPU	18.3	-	-	-	-	-
8CPU(4+4)	12.5	99.0	18.5	16.2	16.1	20.9

表 9：テストプログラムの実行時間[sec]
Table9：Execution time of test-program

ホームノード数	1	2	4
実行時間	1.08	0.638	0.607

LU では、上記オーバヘッドを除去したMH方式の最適な場合についても、性能劣化がみられる。この原因として、1) ノード数不足でMH方式の効果が現れない 2) LU は、バリア同期区間に Producer がおこなうデータ生成と計算が同居しており、IS にようにバリア同期とライトバックがオーバラップしない。3) 上述の通りホームノード以外での書き込みがないため、MM のようにMH方式で新たに発生するライトバックがSH方式のライトバックと一括して転送されることもない。この組み合わせ的要因により、MH方式による新たなライトバックがSH方式の場合に対して新たな逐次処理部分を形成してしまっており、速度低下してしまう。

異なる問題サイズを用いたり、並列度が上がった場合、ページアライメントに合わないデータ配置がおこりうる。このような場合については、SH方式でライトバックが引き起こされるため、MH方式の新たなライトバックもこれと一括処理することができ、速度向上が期待できる。

また、同時に、MH方式に加えて、MH方式以外のクラスタキャッシング方式であるプロキシホーム方式を用いることでさらなる速度向上が得られることを示唆する結果でもある。3.2 でも述べたように、プロキシホーム方式では、On-the-fly 的にページをキャッシュするため、MH方式のように確実にクラスタ間通信路を介するページ授受がなくなるわけではない。しかし、プロキシホーム方式のオーバヘッドはないことはないが、MH方式のようにプレロード的におこなうライトバックがその他処理とオーバラップできずに新たなオーバヘッドになることは少なくともないと考えられる。本予備評価で用いた LU などに関しては、プロキシホーム方式で速度向上が期待し得る。

5.3.4 クラスタ性能に固有な実装方法の評価

4.3 でも述べたように、本稿では多彩なマルチクラスタ環境に適応したホームノードの数が算出されるテストプログラムを作成した。今回は Producer-Consumers パターンに着目した。テストプログラムは、Producer がホームノードとなっている複数ページに書き込みをおこなう。その後、Producer 以外のノードで複数ページを連続的に読み出す、簡単なものである。実験的に、パラメータを、ページサイズ 4K、ページ数は 100 ページとした。また、計算ノード数は 16、ホームノードの数を1, 2, 4として測定した。同プログラムの実行結果を表9に示す。

表9から、今回作成したテストプログラムでは、ホームノードの数を1, 2, 4として測定した結果、最適なホームノードの数は4となった。しかし、5.3.1 で示されている通り、MM については、ホームノードの適切な数は2となることが望ましく、テストプログラムの精度が低いことを示す。テストプログラムのように Consumers が立て続けにページリクエストを

だす場合もあれば、そうでない場合もある。それによってホームノードのページ読み出しのコンテンションが変化するため、SCあるいはMCの各要素クラスタにおけるホームノードの数に影響を与える。また、ページ数はMC間でホームノードを複数にするかに大きな影響を与える。

このように、クラスタごとのホームノードは、例えばMCにおいては、片方のクラスタでは2つ、もう片方では1つともなり得る。

従って今後は、テストプログラムの精度を上げるために、入力するパラメータの項目を増やし、様々なアプリケーションプログラムに対応できるプログラムを作成していく。

5.3.5 マルチクラスタ向けバリア同期の実装の評価

4.5 でも述べたように、本稿ではMC向けのバリア同期の評価をおこなった。そのためにまず、実行再開始通知をクラスタ内ノード優先とクラスタ外ノード優先で送信したときのバリア同期単体の時間を測定した。測定はバリア同期を 100 回実行してその平均値を求めることでおこなった。表 10 に測定結果を示す。

表 10 より、バリア同期成立確認後に送信する実行再開始通知は、これを送信するノードとは異なるクラスタに属するノードから転送したほうが効率がよいことが分かった。

次に、実際のアプリケーションプログラムを実行した場合での影響をみる。アプリケーションプログラムには前述のテストプログラムを使用し、実行再開始通知を 1)クラスタ内優先に、2)クラスタ外優先に、3)まず Producer に送信した後、Producer 以外のノードに対してクラスタ外優先に送信する producer 優先の 3 つの場合について、MH方式での実行時間を測定した結果を表 11 に示す。その際、実行再開始通知を出すノードをノード 0、Producer をノード 1, ノード数を 16, ページ数を 1 とし、100 回繰り返した場合のものである。

表 11 より、さらに 4.5 で述べたように、Producer のライトバックと Consumers の読み出しをオーバラップさせようと試みた結果、性能向上が得られた。また今後は、各ノードの計算量なども考慮に入れて、どのノードから実行再開始通知を送信するべきなのか、検討していきたい。

6. 関連研究

[5]では、diff 分散方式を採用した TreadMarks を SDSM として用いてMC環境に向けて同様の評価実験をおこなっているが、速度向上を示すには至っていない。我々の提案方式は我々が継続している大規模クラスタシステムに向けられており、大規模クラスタシステムをいくつのサブクラスタとみたらよいかというクラスタ構成法的な観点が[5]と異なる点である。このようなアプローチで JIAJIA では得られなかった速度向上とスケーラビリティを低速なネットワークでも実現した。

また、本稿では、共通のアプローチでMCと大規模クラスタシステムを扱っているため、例えば大規模クラスタシステム同士をつなげた大規模MC上で、組み合わせ的に提案方式が透過的に適用可能となるのが提案方式の最大の特徴である。

7. おわりに

本稿では 2 種類のMCにおいてクラスタごとのホームノードのクラスタキャッシュとしての有効性を示した。現在は、さらに効率的なクラスタキャッシング方式であるプロキシホーム方式の実装を進めている。

また、クラスタ間通信遅延を隠蔽するために有効であると考えられる一貫性粒度あるいは通信粒度を調整する機能を追加することもあわせて検討している。

表 10：バリア同期の実行時間 [msec]
Table10：Barrier synchronization time

	クラスタ内優先	クラスタ外優先
バリア同期区間	5.79	5.32

表 11：バリア同期の実装方法の違いによる実行時間
Table11：Execution time with different implementations of barrier synchronization

	クラスタ内優先	クラスタ外優先	producer 優先
テストプログラム	2.26	2.23	2.16

本稿の評価において、各データに対するホームノードの配置は最適であることを前提としてきた。ホームベース方式の SDSM では、とくに大規模クラスタシステムにおいて、このデータマッピングが大きく性能に影響を及ぼす[2]。ホームノードの配置を最適にするには、ファーストタッチでおこなう方法、ホームノードの動的再配置[7]でおこなう方法、ユーザが明示的に指定する方法やこれらの組み合わせが考えられるが、これも今後の重要な課題である。

本稿では、低速なネットワーク環境でも高性能が得られることが明らかになったが、クラスタ専用高速ネットワークである Myrinet に関しても近くデータをとる予定である。また、SCore システムソフトウェア上で動作する SDSM SCASH 上でも実装をおこない、ネットワーク性能や SDSM の一貫性制御方式・実装方式が異なる環境での性能評価や、ホームノードの最適な数などのクラスタ構成法的アプローチの検討も今後の課題としたい。

また、MC環境や大規模クラスタシステムでは、SDSM の一貫性維持機構などによるトラフィックのコンテンションを回避することが重要であると考える。例えば、制御用の通信が、データ転送用の通信でブロックされてしまうことがあり得る。このような場合、現在区別されていないこれらを分別し、優先度制御をおこなうことなどが重要となってくると考えられる。現在、そのための環境の構築を進めている。

MH方式は、SDSM に限って有効な方式ではなく、大規模クラスタシステムやMCあるいは大規模クラスタシステムのMCなどで、あらゆる並列プログラミングライブラリで適用可能であると考えられる。

参考文献

1) 工藤知宏ほか: AIST スーパークラスタ構想, 情報処理学会研究報告, Vol. 2002, No. 91, pp. 103-106 (2002).
2) 佐藤三久ほか: Cluster-enabled OpenMP: ソフトウェア分散共有メモリシステム SCASH 上の OpenMP コンパイラ, 情報処理学会論文誌, Vol. 42, No. SIG 9 (HPS 3) , pp. 158-159 (2001).
3) Hu, W. et al: JIAJIA: A Software DSM System Based on a New Cache Coherence Protocol, HPCN Europe, pp. 463-472 (1999).
4) Keleher, P. et al: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, Proc. of the Winter 1994 USENIX Conference, pp. 115-131 (1994).
5) Arantes,L. et al: The Impact of Caching in a Loosely –coupled Clustered Software DSM System, Proceedings of the IEEE International Conference on Cluster Computing, pp. 27-34 (2000).
6) 城田祐介ほか: ホームベースソフトウェア分散共有メモリ上で Migratory Access を効率良く処理する権限委譲プロトコル, 情報処理学会論文誌, Vol.44, No.1, pp.103-113(2002).
7) 原田浩ほか：ソフトウェア分散共有メモリ SCASH におけるページ管理ノードの動的再配置機構の実装と評価, 情報処理学会研究報告, Vol.1999,No.77,pp.89-94(1999).