ハイパフォーマンス
コンピューティング 77−16
（1999. 8. 3）

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

原田 浩[†] 石川 裕[†] 堀 敦史[†]
手塚宏史[†] 住元真司[†] 高橋俊行[†]

既存の Unix オペレーティングシステムと低通信遅延かつ高通信バンド幅を有するネットワークシステム上に SCASH と呼ぶソフトウェア分散共有メモリを開発している。SCASH 上に、ページ管理ノードの動的再配置機構を実装し、SPLASH2 の LU を用いて評価を行った。その結果、管理ノードをノードに固定的にラウンドロビンに配置した場合との比較では 64 台の実行で 1.17 倍の性能向上を達成したが、管理ノードを固定的に最適化した場合との比較では、0.722 倍の性能を得るに留まった。

Implementation and Evaluation of dynamic page manager node reallocattion mechanism on SCASH Software Distributed Shared Memory

Hiroshi HARADA ,[†] Yutaka ISHIKAWA ,[†] Atsushi HORI ,[†]
Hiroshi TEZUKA ,[†] Shinji SUMIMOTO [†]
and Toshiyuki TAKAHASHI[†]

We have been developing a software distributed shared memory system called SCASH on top of a Unix with a low latency and high bandwidth network system. A dynamic owner node reallocation mechanism is introduced to SCASH and evaluated using the LU benchmark from SPLASH2 suits. LU under the dynamic ower node reallocation mechanism is 1.17 times faster than that of using round robin fashion node allocation. It is 0.722 times faster than that of using statically optimized owner node.

1. はじめに

我々はギガビットネットワークの一つである Myrinet[1] 上に低通信遅延かつ高通信帯域幅を提供する高速通信ライブラリ PM[2),3)] を用いて、オペレーティングシステムのメモリ管理機能を利用した SCASH[4),5)] と呼ぶソフトウェア分散共有メモリを実現している。

使用しているプラットフォームは 128 台の Intel PentiumPro プロセッサ (200MHz) と Myricom Myrinet ネットワークから構成される PC クラスタで、各ノードプロセッサのカーネルとして Linux[6)] を使用している。

SCASH は、これまで共有メモリ領域内の各ページを管理するノードを静的に配置してきた。しかし、管理ノードを静的に配置した場合、ユーザがアプリケーション実行前に、最適な管理ノードを明示的に指定しなければ、ページコピー等のオーバーヘッドが障害となり充分な台数効果を得ることは困難である。

そこで、管理ノードをアプリケーションの実行中に自動的に再配置すれば、ユーザが明示的に管理ノードを指定しなくとも充分な台数効果を得る事が期待できる。

SCASH にページ管理ノードをプログラムの実行中に動的に再配置する機構を新たに加え、実際にアプリケーションを用いて評価を行う。はじめに節 2で SCASH の概要とページ管理機構について説明する。次に、節 3でページ管理ノードの動的再配置機構について説明する。節 4と 5において SPLASH2 の LU を用いた

評価方法と評価結果を述べ、節 refsec:discuss において評価結果に関して検討を行う。節 7において関連研究を紹介し、最後に節 8でまとめを行う。

2. SCASH

2.1 SCASH の概要

SCASH は、Myrinet[1] 上に低通信遅延かつ高通信

† 新情報処理開発機構つくば研究センタ
Tsukuba Research Center, Real World Computing Partnership

帯域幅を提供する高速通信ライブラリ PM[2),3)]を用いたソフトウェア分散共有メモリである。オペレーティングシステムのメモリ管理機能を利用し、ユーザレベルのライブラリとして実現されている。

共有メモリ領域の一貫性維持は、オペレーティングシステムが提供するページ単位で行われる。一貫性モデルとして ERC(Eager Release Consistency)[7),8)]を採用し、その実装にはマルチプルライタプロトコル[9)]を用いている。さらに、ページ単位の一貫性維持プロトコルとして、ページの無効化を通知する無効化プロトコルと、ページデータを送信し、ページの更新を通知する更新プロトコルの双方を実装し、実行時に選択できる。

以下に SCASH が提供している機能を示す。

- 共有メモリの初期化、割り当て、開放
  共有メモリを全ノード上で、割り当て、開放を行う。SCASH は共有メモリを全ノード上で等しいメモリ空間に割り当てる。共有メモリを割り当てるメモリアドレスは、初期化時に指定可能である。
- 同期機構
  SCASH はメモリバリアとロックの 2 つの同期機構を提供している。メモリバリアは、全ノードでバリア同期を取り、共有メモリの全内容が、全ノード上で同一の最新内容に更新されることを保証する同期機構である。ロックはブロッキングロックが提供される。ロックは分散ロックキューによって実現されている。
- 一貫性制御機構
  SCASH では、書き込み共有メモリデータのホームノードへの書き戻し、ホームノードからの読み込みなどの、一貫性維持機構の一部をライブラリとしてユーザに開放している。
- その他
  SCASH は、グローバルメモリのデータをブロードキャストする機能、ページ単位で管理ノードを指定するなどの機能を提供している。特に、あるノードに局所的な共有メモリ領域に関しては、ノードを管理ノードとして指定することによって、通信量を削減し実行性能を向上させることができる。

現在 SCASH は、新情報処理開発機構の PC クラスタ 2 号機上で稼働している。PC クラスタのノード数は 128 台であるが、ページディレクトリの制限から、SCASH の実行ノード数は最大 64 台である。

2.2 SCASH のページ管理機構

SCASH では共有メモリ領域はページ毎に管理される。各ノードは共有メモリ領域のページテーブルを保持している。ページテーブルには、各ページの保護状態と以下に示すホーム、オーナーと呼ばれるページを管理するノードを記録している。各ノードは、常に正しいホームノードを知っているが、常に正しいオーナーは知らない。但し、ホームノードは常に正しいオーナーを知っている。共有メモリ領域の全てのページは、必ず一つのホームとオーナーを持つ。ホームとオーナーはそれぞれ以下のように各ページを管理している。

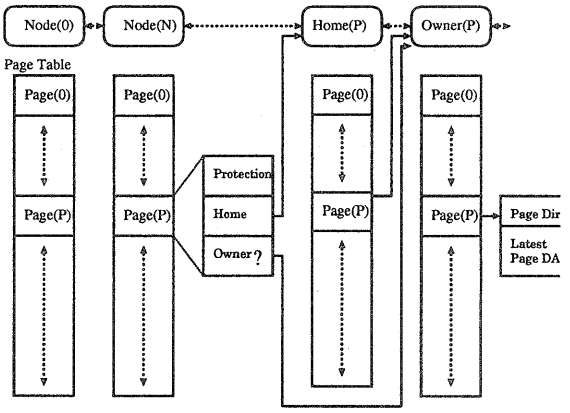

図 1　ページの管理

- ホーム
  最新のオーナーノードを常に記録しているノード。全てのノードは、全ページのホームノードを記憶している。最新のオーナーが不明になったノードは、ホームにオーナーを問い合わせる事によって、最新のオーナーを得る事が出来る。
- オーナー
  ページの最新データと、ページを共有しているノードの集合 (以下ページディレクトリと記述する) を保持しているノード。ページの共有に参加するノードは、ページの最新データをオーナーからコピーする。またバリア同期を取るさい、ページデータを更新したノードは、元のページデータとの差分をとり、オーナーに通知する。オーナーは、各ノードから受信した全ての差分をページに適用する事により、常に最新のページデータを保持する事ができる。

SCASH では、共有メモリをアロケートした直後は、ホームとオーナーは等しいノードに設定される。また、ホームとオーナーは、各ノードに均等に分散するように配置される。

3. ページ管理ノードの動的再配置

これまで述べてきたように、SCASH のページ管理機構では、オーナーが常にページの最新データを保持し、ページの共有に参加するノードは、オーナーからページをコピーし、更新データ (元のページデータとの差分) をオーナーに送信する。そのため、各ページのオーナーの配置によって、ページコピー、差分の送信によるコストが大きく異なる。そこで SCASH で

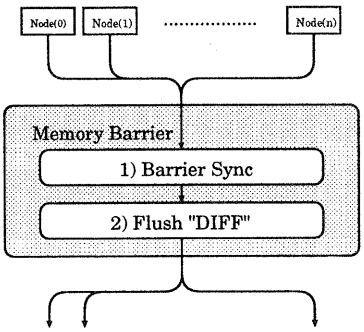

図2　オーナー固定の場合のバリア実行手順

は、各ページのオーナーをプログラム実行中に動的に移動、再配置する事により、ページコピー、差分送信のコスト削減を計ることにした。

3.1 再配置の手順

SCASHは、オーナーの再配置を行わない場合、図2に示すように以下の手順でバリア同期を実行する。

(1) Barrier Sync
最初に全ノード間でバリア同期を取り、全ノードが、共有メモリへのアクセスを中断し、メモリバリアの実行を開始したことを確認する。

(2) Flush DIFF
全ノードで、書き込みが行われた全ページについて、書き込みが行われる以前のページと書き込みが行われたページを比較して、差分を作成し、該当ページのオーナーへ送信する。差分を送信した後、ページのプロテクションを読み書き可から読み込み可に変更する。ページのプロテクションを読み込み可に変更するのは、ページに対する書き込みアクセスによってページ例外処理ルーチンを起動させるためである。起動されたページ例外処理ルーチンは、ページの複製を作成し、ページプロテクションを読み書き可に戻す。差分を受信したオーナーは、該当ページを共有している全ノードに対して無効化メッセージを送信し、ページを無効化する。全ての差分の送信が終了した時点でバリア同期を取る。

SCASHでは上記のように、バリア同期によって共有メモリデータの同期を取るため、各ページの差分がオーナーに集められる。そこでSCASHでは、バリア同期の実行手順を以下のようにする事によって、バリア同期の実行時に各ページのオーナーノードを再配置する事にした。

(1) Barrier Sync
最初に全ノード間でバリア同期を取り、全ノードが、共有メモリへのアクセスを中断し、メモリバリアの実行を開始したことを確認する。

(2) Flush DIFF

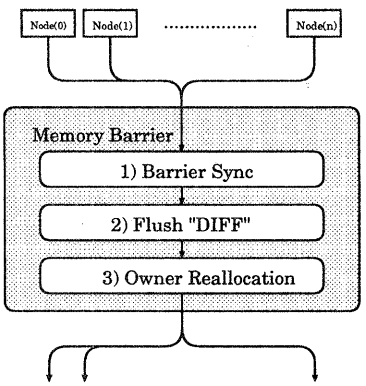

図3　オーナーを再配置を行うバリア実行手順

全ノードで、書き込みが行われた全ページについて、差分を作成し該当ページのオーナーへ送信する。差分を送信した後、ページのプロテクションを読み書き可から読み込み可に変更する。全ての差分の送信が終了した時点でバリア同期を取る。

(3) Owner Reallocation
差分を受信したオーナーは、差分を送信したノード、差分の大きさからバリア同期終了後のオーナーノードを決定する。オーナーの決定の詳細は次節で述べる。オーナーノードが変更される場合、新しいオーナーにページをコピーし、ホームに新しいオーナーを通知する。また、ページを共有していた全ノードに対して、インバリデートメッセージを送信する。インバリデートメッセージには新しいオーナーも含まれる。すなわちページを共有していたノードは、インバリデートメッセージを受信する事によって常に最新のオーナーを知ることができる。

3.2 オーナーの選択

SCASHでは、以上のような手順によりバリア同期実行時に、オーナーの再配置を行う。オーナーは、ページを更新してもバリア同期実行時に差分の生成、送信を行う必要がない。ページに対して読み込みを行った場合も、ページを転送する必要がない。

オーナーの再配置を行う場合、次回のバリア同期までに、各ページに対して最も大きな更新を行うノードがそのページの新たなオーナーとして選択されるのが理想である。

しかし、未来のメモリ書き込みを正確に把握する事は不可能である。SCASHでは、以下の2つを仮定してオーナーの選択を行う事にする。

- バリア同期までに書き込みをしたノードは、次回のバリア同期までに、書き込みをする可能性が高い
- 次回のバリア同期までに書き込みを行う可能性は、

書き込みの量が大きい程高い

各ノードの書き込みの量は、差分の大きさである。すなわち、オーナーは、受信した差分の大きさを比較し、最も大きな差分を送信して来たノードを新たなオーナーとして選択するべきである。但し、SCASH の現在の実装では、オーナーの移動にはページコピーが伴う。そこで、オーナーが書き込みを行っている場合は、引き続きオーナーが書き込みを行う可能性が高いと予想して、オーナーの移動は行わない事にする。以上から、SCASH におけるオーナーの選択のアルゴリズムは以下のようになる。

- オーナー選択のアルゴリズム
  オーナーがページに対して書き込みを行わず、オーナー以外の共有ノードが書き込みを行った場合のみ、書き込み量の最も大きなノードにオーナーを移動する。

4. 評価方法

4.1 アプリケーションによる評価

測定には SPLASH2[10] から LU を用いる。LU はバリア同期のみで記述されているので、メモリバリアとページ例外処理以外のオーバーヘッドは存在しない。アプリケーションの実行性能を把握するため、アプリケーション実行時間とアプリケーション実行時間中のバリアの実行時間を計測する。また、アプリケーション実行中の通信量として、ページ共有のためのページコピー回数、ページデータ更新のための差分の送信回数、オーナーの移動回数、そして、ホームに対するオーナーの問い合わせ回数を測定する。

測定に用いる問題サイズは表1の通りである。

表1　LU の問題サイズ

Matrix	2048 x 2048
Block	64 x 64

以下の通り3種類のオーナーの配置方法について実行時間と通信量を測定する。

(1) ラウンドロビン
ホーム、オーナー共、SCASH のデフォルトであるラウンドロビンに各ノードに割り当てる。オーナーはプログラム実行中に移動しない。

(2) オーナーを動的再配置
これまで述べた手法により、アプリケーション実行中にオーナーを動的に再配置する。

(3) ホームを最適化
SCASH システムの初期化後、アプリケーションがホームを明示的に指定する。アプリケーション実行中、各ページに対して最も書き込み回数と読み込み回数の多いノードがホームとして指定される。オーナーはプログラム実行中に移動しない。

ノード数を1台から64台で動作させた場合の実行時間を測定する。ノードが1台の実行時間は、SPLASH2 の nullmacro を用いた実行時間とする。ノード数1台の実行では、SCASH のライブラリは呼び出されないので、SCASH のオーバーヘッドは存在しない。

プログラムの測定は全て 10 回行い、平均値を採用する。

4.2 評価環境

測定は全て我々が開発した PC クラスタ 2 号機上で行う。PC クラスタ 2 号機の主な仕様を表2に示す。Myrinet 用通信ライブラリ PM の最小遅延時間と最大帯域幅はそれぞれ、7.2μsec、117.6 MBytes/sec である。

表2　PC クラスタの主な仕様

# of Node	128
CPU	Intel PentiumPro 200Mhz
Cache	512KBytes
Chipset	440FX
Memory	EDO 256MBytes/Node
Network	Myrinet 1.28GBits/sec
Node OS	Linux 2.1.119

5. 評価結果

5.1 実行時間

LU の実行時間とメモリバリアの実行時間を図4に示す。各実行ノード数ごとに縦棒が3本並んでいるが、これは左から順に、オーナーの配置がラウンドロビン(デフォルト)、オーナーを動的に再配置、最適なホームを指定した場合の実行時間を示している。

縦棒は、それぞれが LU 全体の実行時間を示している。うち下の部分が、全体の実行時間に占めるメモリバリアの実行時間である。

5.2 通信量

つぎにアプリケーション実行中に行われた、ページコピー、差分の送信、オーナーの移動、オーナーの問い合わせの回数を表3から表5に示す。表3、表5はオーナーの配置が固定であるため、オーナーの移動、オーナーの問い合わせは発生しない。

表3　LU の通信量 (オーナーラウンドロビン)

# of Node	ページ	差分
2	47736	45636
4	74754	68454
8	94507	79863
16	109032	85544
32	128267	88366
64	146205	89860

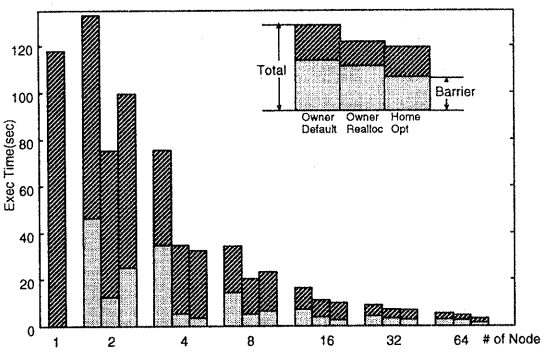

図4　LU の実行時間

表4　LU の通信量 (オーナー再配置)

# of Node	ページ	差分	オーナー移動	オーナー問い合わせ
2	6260	6076	4028	9864
4	9968	7614	6078	15738
8	17704	8023	7127	24649
16	25664	8148	7668	33238
32	41744	8184	7928	49632
64	57918	8190	8064	65944

表5　LU の通信量 (ホームを最適化)

# of Node	ページ	差分
2	47736	45636
4	8432	0
8	60308	45636
16	25184	0
32	84900	45636
64	57792	0

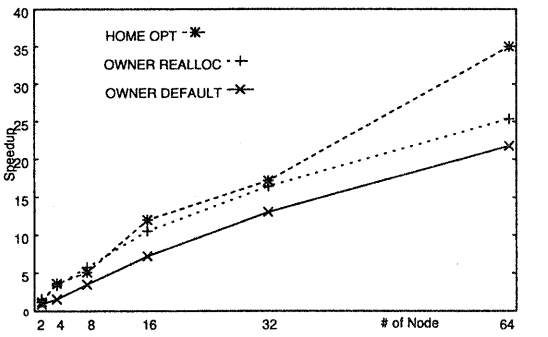

図5　台数効果

5.3 台数効果

測定結果から得られたLU の台数効果を図5に示す。

6. 検　　討

6.1 LU

オーナーを動的に再配置した場合、64 ノードの実行で 25.4 倍の台数効果を得る事ができた。これは、オーナーをラウンドロビンに固定配置した場合より、1.17 倍高速であるが、ホームを最適化した場合と比較すると、0.722 倍の性能に留まっている。

図4から、ノード数が8台までは、オーナーを動的に再配置する事によって最も高い性能を得られるが、実行ノード数が増えるに従い、ホームを最適化した場合の実行性能が高くなる傾向がわかる。

オーナーを動的に再配置した場合と、最適なホームを指定した場合の実行時間を比較すると、実行時間の差は主にバリア同期の実行時間の差である事がわかる。台数が増加するにつれて、オーナーを動的に再配置した場合よりもホームを最適化した場合のバリアの実行時間が、小さい事がわかる。

通信量を比較すると、オーナーを動的に再配置した場合、全ての実行ノード数において、オーナーをラウンドロビンに固定した場合よりもページのコピー数、差分の送信数ともに減少している事がわかる。但し、ノード数が増加するにつれて、ページのコピー数、差分の送信数ともに増加傾向にあることがわかる。またオーナーの移動数、ホームへのオーナーの問い合わせ回数も、ノード数が増加に従い増加する傾向にあることがわかる。

7. 関連研究

SCASH では、管理ノードを動的に再配置する事によって、パフォーマンス向上を狙った。論文[11]では特定のノードに資源の管理を集中させない Symmetry なプロトコルと特定のノードに資源を管理させる asymmetric なプロトコルを比較、検討した上で、管理ノードが動的に移動するプロトコルを実装し、評価を行っている。

8. まとめと課題

最新の高速ネットワーク技術を用いて、SCASH と呼ぶソフトウェア分散共有メモリを PC クラスタ上に開発している。

SCASH 上に各ページのオーナーをアプリケーション実行時に動的に再配置する機構を実装し、SPLASH2 から LU を用いて測定、評価を行った。オーナーを各ページ毎にラウンドロビンに静的に割り当てた場合と比較して 1.17 倍の性能向上を計ることができた。

但し、ホームノードを最適に割り当てた場合との比較では、0.722 倍の性能に留まることが解った。

8.1 オーナーの動的再配置に関して

オーナーノードの動的再配置機構を実装するにあ

たって、一層の性能向上を計るには、より小さなオーバーヘッドで、いかに最適なオーナーを選択できるかが課題となる。

今回の実装では、書き込みをしたノードだけを対象にオーナーを選択したが、読み込みを行ったノードも新たなオーナーとして検討されるべきである。オーナーが書き込みをしていない場合、オーナー以外のノードの更新した大きさによってのみオーナーを決定しているが、書き込み量の履歴を管理することによってより適したオーナーを選択できる可能性もある。

メモリアクセスパターンが静的なアプリケーションであれば、全ノードの全ての共有メモリ領域へのページアクセスをプロファイルし、その結果から、最適なホームとオーナーを求めることができるはずである。求められた最適なホーム及びオーナーと、動的に求められたオーナーを比較することによって、オーナー選択の精度を求めることができる。以上のようにプロファイリングの手法を用いる事によって、オーナー選択の精度を求め、より正確にオーナーを選択する方法を探る事も今後の課題の一つである。

各ノードは、ページを共有しているページに対しては、インバリデートメッセージを受信する事によって、常に正しいオーナーを保持しているが、新たに、共有に参加するページに関しては、ホームにオーナーを問い合わせなくてはならない。オーナーを問い合わせるコストを削減することも今後の課題として上げられる。

また、今回の実装では、オーナーの移動時にページのコピーを行っているが、新たに選択されたオーナーに対して、差分を送信することによって、ページのコピーを省略できる可能性がある。

近年、逐次プログラムを並列プログラムに書き換える並列化コンパイラの研究が盛んに行われている。これら並列化コンパイラは、データを各ノードへ局所的に配置する事によって、高い台数効果を獲得している事が多い。今回は、プログラム実行中に自動的にオーナーを再配置する機構について実装、評価を行ったが、並列化コンパイラのデータの割り当てに応じてオーナーを動的に再配置する事も今後の課題の一つである。

謝　辞

本研究における実装、評価に関して貴重な御意見を頂いた新情報処理開発機構工藤 知宏氏、山本 淳二氏に感謝いたします。

参 考 文 献

1) http://www.myri.com.
2) Hiroshi Tezuka, Atsushi Hori, Yutaka Ishikawa, and Mitsuhisa Sato. PM: An Operating System Coordinated High Performance Communication Library. In *High-Performance Computing and Networking '97*, 1997.
3) 手塚, 堀, O'Carroll, 原田, 石川. ピンダウンキャッシュを用いたユーザレベルゼロコピー通信. 情報処理学会研究報告. 情報処理学会, August 1997.
4) 原田浩, 手塚宏史, 堀敦史, 住元真司, 高橋俊行, 石川裕. Myrinet を用いた分散共有メモリにおけるメモリバリアの実装と評価. 並列処理シンポジウム JSPP'99, pp. 237–244. 情報処理学会, June 1999.
5) 原田, 手塚, 堀, 住元, 高橋, 石川. Myrinet を用いた分散共有メモリシステムの評価. ハイパフォーマンスコンピューティング研究会資料, 98-HPC-73, pp. 73–78. 情報処理学会, October 1998.
6) http://www.linux.org.
7) K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A.Gupta,and J.Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proceedings of the 17th Annual Symposium on Computer Archtecture*, pp. 15–26, May 1990.
8) J.B Carter, J.K. Bennett, and W.Zwaenepoel. Implementation and Performance of Munin. In *Proceedings of the Thirteenth Symposium on Operating Systems Principles*, pp. 152–164, October 1991.
9) Amza C. Cox A. L. Dwarkadas, S. and Zwaenepoel W. Software DSM Protocols that Adapt between Single Writer and Multiple Writer. In *Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA-3)*, pp. 261–271, February 1997.
10) Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *In Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 24–36. ACM, June 1995.
11) Peter J. Keleher. Symmetry and performance in consistency protocols. In *In Proc. of 1999 International Conference on SUPERCOMPUTING(ICS'99)*, pp. 43–50, June 1999.