A–02

分散共有メモリを用いたアドレス空間共有方式
Sharing Address Spaces using Distributed Shared Memory Technique

小鍛治 翔太[††]，芝 公仁[†]，岡田 至弘[†]
Shota Kokaji[††]，Masahito Shiba[†]and Yoshihiro Okada[†]

1　はじめに

近年, 計算機の扱うデータは大容量化し, 処理も複雑となっている. それに伴い, 計算機には複数のプロセッサが搭載され, 処理の高速化がはかられている. このような計算機では多くのプロセスとスレッドが同時に実行され, 高い並列実行性を持つが, 1 台の計算機が持つ計算機資源は限られている. そのため, 複数の計算機をネットワークで接続し分散並列処理を行うクラスタリングなどが盛んに行われている.

分散共有メモリはこうした計算機資源の共有を行う手法の一つであり, 計算機間でのメモリの共有を実現する. これにより, プロセスは計算機間でデータを共有することが可能になる. しかし, プロセスの持つアドレス空間が共有されるわけではなく, データのみの共有であるため, プロセスコードの共有には対応することができない.

本研究では, プロセスのもつ資源として最も重要なものの一つである, アドレス空間の共有を実現するシステムを構築した. これにより単一のプロセスを複数の計算機上にまたがって動作させることができ, そのプロセスが持つスレッドがそれぞれの計算機の資源を効率的に使用できるようになる. 本システムでは次の 3 つの特徴を持つ.

- アドレス空間の共有が可能である
- 並列処理用にコードを書き直す必要がなく, 既存のコードを用いることができる
- 専用のハードウェアを使わず, ソフトウェアのみで実現している

アドレス空間の共有により, 複数の計算機でプロセスの持つスレッドを実行することができるため, システム全体で単位時間あたりの処理できるデータ量が大きくなる. また, 複数のスレッドを用るプログラムであれば, コードを分散処理用に記述する必要がない. さらに, 本システムはソフトウェアのみで実現するため, 導入のコストが低いという利点がある.

本システムの構築において, アドレス空間全体を共有するためのシステムコールと分散共有メモリの一貫性を維持するための機構を作成した. 本システムは, 現在, Linux カーネル 2.6.33 上に構築されている. 本システムはプロセスを共有するため, 共有するプロセスのアドレス空間を他の計算機にコピーし, また, アドレス空間への書き込みが発生した場合, 読み出すメモリ内容に違いが出ないようその内容を他の計算機に知らせている. このような複数の計算機で同じメモリ内容を維持させる一貫性制御のため, 共有するアドレス空間をページ単位で管理し, 更新されたページの内容を取得した計算機は共有するプロセスのアドレス空間に取得したページ内容の書き込む. また, ページ内容を送信する際にも, アドレス空間からページ内容の読み出しを行っているが, Linux カーネルが持つシステムコールでは 4 バイト単位でしか他のプロセスのメモリにアクセスできない. そのため, ページ内容のコピーに時間がかかる. 本稿では, ページ内容のコピー時間を短縮するために追加したシステムコール, および，それを用いたメモリの一貫性制御処理の評価について述べる. 以下, 本稿では, 第 2 章で関連研究について述べ, 第 3 章で本システムの構成とシステムが持つ機能, 第 4 章でアドレス空間の共有を実現するメモリの一貫性制御について述べる. また, 第 5 章で本システムの評価を行う.

2　関連研究

分散共有メモリでは一貫性制御の処理速度が重要な要素であり，通信の高速化を図るために Ethernet に代わり InfiniBand を用いたシステムが提案されている [1]. ここで提案されているシステムでは, InfiniBand の利用により, データ転送性能を高め, 一貫性制御に必要なデータ交換を高速化している. しかし, 専用のネットワーク機器を使用しているため, 導入コスト

[††] 龍谷大学大学院理工学研究科
Graduate School of Science and Technology, Ryukoku University
[†]龍谷大学理工学部
Faculty of Science and Technology, Ryukoku University

が高くなる. また, 一度に送れる最大データサイズがページサイズより小さく, 必ずしも分散共有メモリに適したネットワーク機器であるとはいえない

コピー時間を短縮する方法としてはゼロコピー通信などがある. 既存の手法ではデータ受信時にゼロコピー通信の実現は難しいが, 門らはこのデータ受信時のゼロコピーの手法を提案している [2]. これにより, データ送受信時の処理は大きく削減されているが, 受信したデータを直接プロセスのアドレス空間内に書き込むことは，セキュリティ上問題となる場合がある.

64 ビットオペレーティングシステムの普及に伴い, 巨大なアドレス空間を有効活用する方法に遠隔計算機の物理メモリを利用するシステムがある. そのようなシステムでは本システムとは違い, 分散処理ではなく逐次処理を行うような並列化が困難なアプリケーションに対応したシステムであるが, 近年多くのアプリケーションがマルチプロセッサに対応したマルチスレッドで作られているため, 利用が難しい. また, これらのシステムも本システムと同様に遠隔計算機からページ内容を送信する必要があり, ページ置換アルゴリズムを用いることで参照の多いデータをキャッシュ上に残し, 転送データへのアクセスを高速化している. 斉藤らはこのようなページ置換アルゴリズムをユーザレベルで実装し評価を行っている [3]. ユーザレベルで実装しているため, 本システムでも使用可能であり, アクセス頻度の高いメモリをキャッシュ上に置いておくことで, システムの効率化がはかれる.

3　システム構成

複数のスレッドを使用するプログラムが多く利用されるようになってきている. 通常, プロセスは単一の計算機上でしか動作することができない. そのため, プロセスは, 複数のスレッドを持っていても他の計算機の CPU 資源を使用することができない.

本研究の目的は, 単一のプロセスを複数の計算機上にまたがって動作できるようにし, そのプロセスが持つスレッドがそれぞれの計算機の資源を効率的に活用できるようにすることである. 計算機間でプロセスを共有するために, 本手法では, ソフトウェア分散共有メモリを実現し, ネットワーク上の複数の計算機でアドレス空間の共有を実現する. 本システムを x86 アーキテクチャ上の Linux カーネル 2.6.33 上に構築し, システムの構成を図 1 に示す.

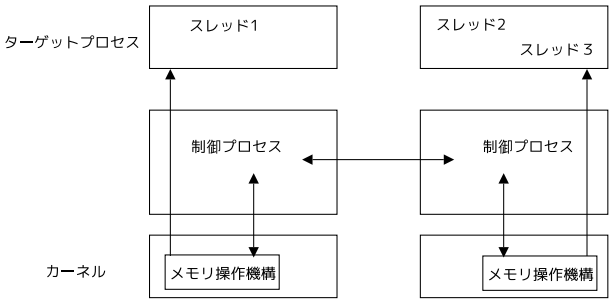

図 1　システム構成

ターゲットプロセスは計算機間で共有するプロセスを指し, ターゲットプロセスの持つスレッドは, アドレス空間を共有する任意の計算機上で動作可能である. アドレス空間の共有を行い一貫性を維持するシステムである制御プロセスは, メモリ操作機構を使用し, ターゲットプロセスのアドレス空間を操作する. また, 他の計算機上の制御プロセスと通信を行い, ターゲットプロセスが必要とするデータを別計算機の制御プロセスから取得する.

3.1　ターゲットプロセス

ターゲットプロセスは本システム上で動作するアプリケーションである. どのようなアプリケーションであれ, 本システム用にコードを書き換えることなく動作させることが可能で, プロセスは本システム上で分散処理されることを意識する必要なく実行できる.

3.2　制御プロセス

制御プロセスは, ターゲットプロセスの状態を常に監視し, 必要に応じて別の計算機上の制御プロセスと通信を行い, ターゲットプロセスが複数の計算機上で実行されているのを意識することなく分散処理を行えるようにサポートするプロセスである. 制御プロセスが実現する機能は主に次の 3 つである.

- 一貫性制御 … メモリの一貫性を維持.
- スレッド移送 … 実行しているスレッドを一時中断し, 別の計算機で実行.
- 入出力制御 … 指定した計算機で実行.

制御プロセスは, ある計算機上でメモリに書き込んだ値がすべての計算機から読み出せるように，メモリの一貫性制御を行う. 制御プロセスは, 生成したスレッドを負荷分散のために動的にスレッドを別の計算機上に移送させる. 移送先は負荷があまりかかっていない計算機を制御プロセスによって動的に選ばれる. ターゲットプロセスが行う入出力処理は

スレッドが動く計算機上で行われるのではなく, ターゲットプロセスが最初に起動した計算機のデバイスで動作が行われる.

制御プロセスはこれらの機能を提供するため, カーネルが持つメモリ操作機構を用いている. これは制御プロセスがユーザプロセスであるため, カーネルの持つ情報へアクセスができないためである. 制御プロセスがユーザプロセスであることにより, 任意のアプリケーションのみを共有させられ点や, システムクラッシュ時の耐障害性といった点でカーネルに機能を組み込むよりも柔軟なシステムとなる.

3.3 メモリ操作機構

カーネル内にあるメモリ操作機構は, 制御プロセスがターゲットプロセスのアドレス空間を操作するためのシステムコールを提供する. メモリ操作機構が提供するシステムコールは以下の通りである.

- p_mmap(pid, addr, len)
- p_munmap(pid, addr, len)
- p_mprotect(pid, addr, len, prot)

p_mmap はプロセス識別子 pid で指定したプロセスが持つアドレス空間のアドレス addr から len バイトの領域を使用可能にする. このとき, 当該領域のメモリページには書き込み権も読み出し権も与えられない. p_munmap はプロセス識別子 pid で指定したプロセスが持つアドレス空間のアドレス addr から len バイトの領域を使用不可にする. p_mprotect はプロセス識別子 pid で指定したプロセスが持つアドレス空間のアドレス addr から len バイトの領域のメモリの保護属性を属性 prot に変更する. なお, addr と len はページサイズの整数倍でなければならない. ページの保護に違反するメモリアクセスを行おうとするとページフォールトが発生し, カーネルはこれをメモリアクセス違反のシグナルとしてプロセスに通知する. 制御プロセスは, ターゲットプロセスへのシグナルを横取りし, ターゲットプロセスのメモリアクセスを検出する.

これらのシステムコールによって, 制御プロセスはターゲットプロセスのアドレス空間を操作することが可能となる. これにより, 制御プロセスは他計算機上の制御プロセスと協調動作することで, アドレス空間の共有が可能となる.

4 制御プロセス

本章では, 制御プロセスが行う一貫性制御の手法と一貫性制御に必要なページ取得処理について述べる. また, 現在実装しているシステムでのページ取得処理の問題点を挙げ, その解決法を示す.

4.1 一貫性制御

各計算機の制御プロセスは, ターゲットプロセスのアドレス空間内のメモリをページ毎に以下の状態を用いて管理する. これらのページの状態は一貫性制御によって遷移する.

- NONE ··· ページの内容を持たず, 読み書きできない.
- READ_ONLY ··· ページ内容を持ち, 読み出せるが書き込めない.
- READ_WRITE ··· ページ内容を持ち, 読み書きできる.

制御プロセスはページの状態を管理し, 他の計算機の制御プロセスと協調し, 各ターゲットプロセスの仮想アドレス空間を順序一貫性 [4] のメモリモデルで一貫性制御を行う. 順序一貫性が実現されるため, ターゲットプロセスはメモリの一貫性制御を意識することなく動作可能である.

制御プロセスのいずれかは各ページの特権状態であるページ管理者となる. ページ管理者は他の計算機で当該ページの状態を変更させたいときにページ状態変更の許可, 不許可を与える権利を持つ. 状態遷移は多くの場合許可されるが, ページ状態が READ_WRITE に状態遷移処理中は許可が与えられず時間をおいて再度要求を送るように通知する. また, ページ管理者はターゲットプロセスがページの書き込み可能になるごとに, 当該ページの書き込み可能な計算機の制御プロセスへと移っていく. 当該ページに書き込み可能な計算機が存在しない場合は, 最後に書き込み可能であった計算機の制御プロセスがページ管理者である. この最後に書き込み可能で, 現在読み出しのみ可能な状態を READ_OWNER とする. READ_OWNER は READ_ONLY と同じく読み出しのみ可能で書き込めない状態であるが, ページ管理者であるため, 他の計算機の制御プロセスからのページの遷移要求を受け付ける.

ページの状態遷移は図 2 のようになる. NONE の状態のページがアクセスされるとページフォールトが起こり, 制御プロセスに通知される. 通知を受け取った制御プロセスはページフォルトが発生したページのページ管理者の制御プロセスに状態遷移の要求を送り, ページ管理者から許可を得て READ_ONLY となる. このとき当該ページのページ管理者はページの内容の複製を渡し, ページの内容を受け取った制御プ

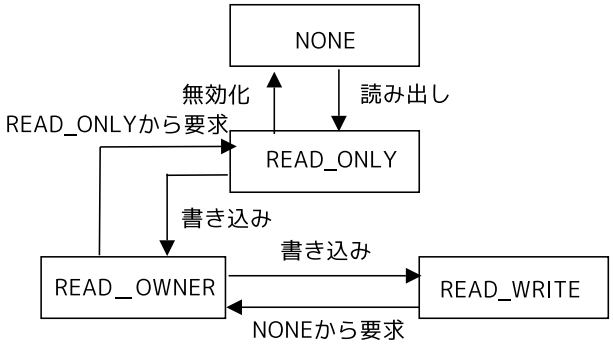

図 2　状態遷移

ロセスはこれをターゲットプロセスのメモリ領域に書き込む. READ_ONLY 時に書き込みを行うとページフォールトが発生し, 当該ページのページ管理者の権利が譲渡され, READ_OWNER となる. また, あるページが READ_ONLY 時に別の計算機の当該ページが READ_WRITE になる場合, READ_ONLY だった計算機の当該ページの状態は NONE に遷移する. あるページの状態が READ_OWNER から READ_WRITE に遷移する前には, 遷移を送る計算機の制御プロセスが共有を行っている全ての計算機の制御プロセスにページの状態を NONE にするように送る. あるページが READ_WRITE の状態を持つ制御プロセスは別計算機から READ_ONLY への遷移要求を受けると当該ページの状態を READ_OWNER へとし, ページのコピーを渡して状態遷移の許可を与える. これらの状態遷移とアクセス保護によってターゲットプロセスの持つメモリの各ページは常に次の 2 つのいずれかを満たす.

- ある計算機のページの状態が READ_WRITE であれば, 他のすべての計算機の当該ページの状態は NONE である.
- ある計算機のページ状態が READ_OWNER であれば, 他のすべての計算機の当該ページの状態は READ_ONLY または NONE である.

これら 2 つの条件により順序一貫性が達成され, ターゲットプロセス内で読まれるメモリの内容は全ての計算機で同一のものとなる.

4.2　ページ取得処理

ある計算機であるページが読み書き可能である場合, 他の計算機では当該ページアクセスすることはできない. このようなページに対してターゲットプロセスが読み出しを行うには, ページ管理者にアクセス許可をもらうと共に, 当該ページのページ内容を取得する必要がある. このページ内容の取得は以下のような手順となる.

(1) ターゲットプロセスが読み出しを行おうとするとセグメンテーションフォルトが発生し, 制御プロセスがこれを取得する.

(2) セグメンテーションフォールトを取得した制御プロセスはページのアクセス権を当該ページのページ管理者である制御プロセスに要求する.

(3) 要求を受け取った制御プロセスは指定されたページを読み出し, 自身のアクセス権を変更を行いページ内容を送信する.

(4) 制御プロセスは受け取ったページ内容をアドレス空間に書き込み, 自身のアクセス権を変更する.

(5) ターゲットプロセスの実行を再開する.

(1) のセグメンテーションフォールトの取得にはシグナルを用いる. Linux ではアクセス権のないアドレスに対してアクセスが起きた場合, セグメンテーション違反を通知するシグナルがカーネルからアクセスを行ったプロセスに通知される. これを制御プロセスが横取りすることで, アクセス権のないページへのアクセスを取得することができる. (3) と (4) では, ページの読み出しと, 送信, 受信, アドレス空間への書き込みにおいてページ内容のコピーが発生する. 具体的にはターゲットプロセスのアドレス空間から制御プロセスへのコピーと送信バッファに書き込むコピー, 送信時のカーネルへのコピー, 受信時のカーネルから制御プロセスへのコピー, 制御プロセスからターゲットプロセスのアドレス空間へ書き込むコピーの計 5 回のコピーである. また, Linux では他のプロセスのアドレス空間へのアクセスに ptrace を用いることができるが, ptrace では 4 バイト単位でアクセスが行われるため, 1 ページ分の読み書きには 1024 回システムコールを発行する必要がある. さらに, アドレス空間からページ内容を読み出す際に, ページ内容が 4 バイトデータの返り値となるため, このデータを送信するには通信バッファに書き込む必要がある. このため, ページ内容を取得する処理に時間がかかる.

このコピー回数を削減し, 全体のコピー時間を短縮するため, Linux に新たにシステムコールを追加した. 追加したシステムコールは以下の 2 つである.

- pokepage(pid, addr, *data)
- peekpage(pid, addr, *data)

表 1　実験環境

CPU	Pentium4 2.8GHz
Memory	512MB
Network	1000Mbps Ethernet

pokepage はプロセス識別子 pid で指定したプロセスが持つアドレス空間のアドレス addr に, アドレス data から 1 ページ分のデータを書き込む. peekpage はプロセス識別子 pid で指定したプロセスが持つアドレス空間のアドレス addr から 1 ページ分のデータを, アドレス data の領域に書き込みを行う. これらのシステムコールにより他のプロセスのページ内容を 1 ページ分アクセスする際, システムコールの呼び出し回数が 1 回になる. また, ptrace では読み出したデータを通信バッファに書き込む必要があったが, peekpage システムコールによって, 読み出したページ内容を通信バッファに直接書き込むことができるため, コピーを削減することができる.

5　評価

本章では, 構築した分散共有メモリのページ取得処理にかかる処理時間を評価する. 本章における性能評価には, 1000Mbps のイーサネットで接続された表 1 の計算機を 2 台を用いた.

5.1　システムコールの処理時間

Linux のシステムコールと追加したシステムコールの処理時間を評価する. どちらもページサイズである 4 キロバイトのメモリの読み出しと書き込みに要した時間を測定した. 読み出しの処理時間はターゲットプロセスから 1 ページ分のページ内容を読み出し, データ送信用の通信バッファにコピーするまでの時間とし, 書き込みの処理時間はバッファに置かれている 1 ページ分のページ内容をターゲットプロセスに書き込む時間とする. そのため, Linux のシステムコール用いた場合は ptrace の 4 バイト単位のアクセスという仕様上, 1024 回のコピーが発生し, また, 読み出しでは ptrace で受け取ったページ内容を memcpy 等で通信バッファにコピーする処理が必要となる. 対して追加したシステムコールを用いる場合では, 一度のシステムコール呼び出しのみとなり, また, 読み出しではページ内容を受け取った領域がそのまま通信バッファとなるためコピーが発生しない. これらの測定結果は 5 回の測定の平均値とする. 実行結果は表 2 のようになった. 表の項目は以下に示す.

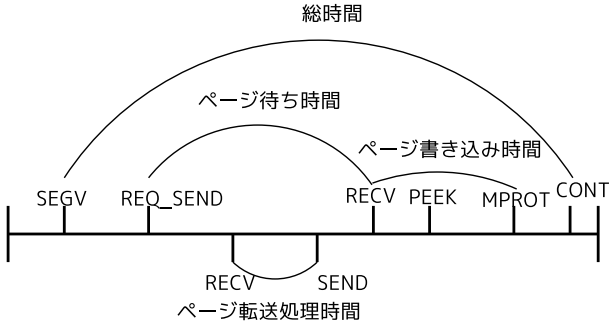

図 3　ページ取得

(A) Linux の ptrace を用いた時の処理時間

(B) 新たに追加したシステムコールを用いた時の処理時間

表 2　システムコール処理時間

	(A)	(B)	速度向上
読み出し	0.870ms	0.003ms	290 倍
書き込み	0.461ms	0.003ms	154 倍

読み出し書き込みともに大きく時間が短縮されていることが分かる. 特に読み出しでは通信バッファへの書き込みがなくなったため, 書き込みの倍近い時間短縮に成功している. また, 新しいシステムコールである pokepage, peekpage の処理時間が同じなのは, どちらもあるアドレスからあるアドレスに同じサイズのデータをコピーするという処理であるためと考えられる. これらのシステムコールを使用することによって, 制御プロセスはメモリの一貫性制御を効率的に行うことが可能になる.

5.2　ページ取得

Linux のシステムコールと新たに追加したシステムコールを用いてページ取得処理にかかる処理時間を評価した. ページ取得処理はアクセスしたページの状態が NONE であれば発生し, ページ内容の送信が起こる. このページ取得処理は前章で述べたような手順となっており, 図 3 はこれらのタイムテーブルである. 時刻を取得するタイミングと処理時間の説明を以下に示す.

- SEGV … セグメンテーション違反を通知するシグナルが発生
- REQ_SEND … ページ管理者にページ内容要求
- REQ_RECV … ページ内容要求取得
- PAGE_SEND … ページ内容送信

表 3　ページ取得時間

	(A)	(B)	速度向上
総時間	1.428ms	0.258ms	5.5 倍
ページ待ち時間	0.664ms	0.213ms	3.1 倍
ページ送信処理時間	0.479ms	0.019ms	25.2 倍
ページ書き込み時間	0.731ms	0.014ms	52.2 倍

- PAGE_RECV … ページ内容取得
- MPROT … ページへのアクセス権変更
- CONT … ターゲットプロセス復帰
- 総時間 … ページ取得に用した全体の処理時間
- ページ待ち時間 … 要求の応答時間
- ページ送信処理時間 … 要求の処理時間
- ページ書き込み時間 … ページ変更時間

REQ_RECV と PAGE_SEND はページ内容を要求する計算機ではなく, 要求ページのページ管理者である計算機での時刻であり, これら以外はすべてページ取得要求を送る計算機での時刻である. 取得した時刻から計算した処理時間を表 3 に示す. なお, 各項目 (A),(B) は前節と同じである.

システムコール導入前に比べて実行時間におよそ 5 倍の差があることが分かる. また, システムコール導入後を見ると, 全体でかかる時間が 0.258ms なのに対して要求を送ってから戻ってくるまでに 0.213ms かかっている. さらに, 要求を受け取ってから返信するのに要する時間は 0.019ms であり, 処理時間の多くが通信に要する時間であることがわかる. この通信時間の割合はシステムコール導入前では 13%程度だったのに対して, 導入後では 75%となっている. このため, これ以上のページ要求処理の時間を削減するには, 独自のプロトコルを用いるなどのサポートをする必要があると考えられる.

6　おわりに

本稿では, 分散共有メモリの一貫性制御における, ページ内容の取得処理に着目し, 新たなシステムコールを追加することでページ内容のコピー時間を短縮する手法について述べた. 本手法では, これまでの手法に比べコピー回数が削減され, またページのコピー時間も大幅に短縮された. これにより, ページ内容の取得待ち状態でいる時間が短縮され, アドレス空間を共有するための処理が効率化された. 今後は, ターゲットプロセスの入出力処理の制御手法や複数の計算機に分散するスレッドのスケジューリング手法などについて検討を行う予定である.

参考文献

[1] 荒木 健志, 斎藤 彰一, 國枝 義敏, 濱太 芳博, 中條 拓伯: InfiniBand Verb 層を利用したソフトウェア分散共有メモリシステム Fagus の実装と評価, , 情報処理学会研究報告. IPSJ SIG Technical Report pp.19-24 (2004)

[2] 門 直史, 田端 利宏, 谷口 秀夫: ゼロコピー通信処理を可能にする実メモリ交換機能の提案, 情報処理学会研究報告. IPSJ SIG Technical Report Vol. 2010-OS-113 No.8 (2010)

[3] 斎藤 和広, 緑川 博子, 甲斐 宗徳: ユーザレベル実装遠隔メモリページングシステムにおけるページ置換アルゴリズムの評価, 情報処理学会研究報告. IPSJ SIG Technical Report Vol. 2010-HPC-125 No.9 (2010)

[4] Lamport, L: How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs, IEEE Trans. Comput. , Vol. C-28, No. 9, pp. 690-691 (1979)