システムソフトウェアと
オペレーティング・システム　89－5
シ　ス　テ　ム　評　価　2－5
（２００２．２．１４）

ソフトウェアDSMの高速化を目的とした通信プロトコルWindの設計と実装

林 章仁[†]　横手　聡[†]　齋藤 彰一[††]　上原 哲太郎[††]　國枝 義敏[††]

[†] 和歌山大学大学院システム工学研究科
[††] 和歌山大学工学部情報通信システム学科

和歌山市栄谷 930 番地

我々は, ソフトウェア DSM(以下 S-DSM) に対応したトランスポートプロトコル“ Wind ”の設計と実装を行った. 現在, Wind は Linux カーネルバージョン 2.4.9 上にトランスポートプロトコルとして実装されている. Wind は, データグラム通信で順序保証と到達保証を行う. 他のトランスポートプロトコルと異なる特徴として, S-DSM に対応した通信処理・メモリアクセスを行うことが可能である. 通信処理では, ユーザプログラムに対し, 同期・非同期通信をプロトコルレベルで処理することができる. 一方, メモリアクセスでは, データ受信時に必要なシステムコールと, ユーザプロセスにスケジュールを切替えるコンテキストスイッチを減少させ, S-DSM の性能向上を図る. 本稿では, Wind の設計・実装方法と性能評価について述べる.

Design and Implementation of the Wind for purposing faster communication of Software-DSM system

Akihito Hayashi [†]　Satoshi Yokote [†]　Shoichi Saito [††]　Tetsutaro Uehara [††]　Yoshitoshi Kunieda [††]

[†]Graduate School of Systems Engineering,Wakayama University
[††]Faculty of Systems Engineering,Wakayama University

930 Sakaedani,Wakayama 640-8510 Japan

This paper deals with a design and its implementation of a new communication protocol,“ Wind ”.Currently Wind is implemented on Linux version 2.4.9 as transport layer protocol.Wind guarantees order and reachable with datagram communication.Wind has two different points compared to exsising transport protocols.These points are on communication and memory access adapting for Software-DSM system.On communication,it is possible to processing both synchronous and asynchronous communicaton to user program.On the other hand in memory access,it decreases a context switches on system.This paper describes its design,its implementation and its performance evaluations of Wind.

1 はじめに

近年, 1 台あたりの PC の高性能化が急速に進んでいる. このような PC を複数台使用し, 大規模演算などの 1 台のマシンでは長時間を要する処理を, 高速に実行するシステムとして, 現在 PC が注目されている. 我々の研究グループでは, PC1 台あたりのコストが安価かつ高性能に成りつつある背景を受け, “ 安価にできる並列計算システムの構築 ”を目指している.

PC クラスタにおいて, 並列計算を行う方法の 1 つとして, ソフトウェア DSM(以下 S-DSM) がある. S-DSM は, ユーザプログラムに対し, 1 つの大きな仮想アドレスを PC クラスタ間で共有できるシステムを提供する. その際, メモリの一貫性を保つためにネットワークを用いたデータ転送を行う.

現在, PC クラスタを構築するために使用するイーサネット, 及び, ギガビットイーサネットは, CPU, 主記憶, 二次記憶などに比べるて速度が低速であることが知られている. PC クラスタ上で S-DSM を用いて計算を行う場合, シュミレーションや解析などの大規模演算が行われることが多い. S-DSM では, 共有するメモリ空間の大きさに比例して, PC 間でのデータ転送が多くなる場合がある. その結果, 全体の処理時間におけるデータ転送時間の占める割合が大きくなる. ゆえに, S-DSM にて大規模演算を行う場合, PC クラスタ間でのデータの受け渡しを効率よく行うことが, S-DSM のパフォーマンスをあげるのに必要である. このデータの受け渡しの効率を向上させ, 高速に処理することを目的としたのが“ Wind ”[1][?] である.

Wind は, ISO が制定した 7 階層のネットワークプロトコルに基づき分類するとトランスポート層にあたる. 現在, トランスポートプロトコルとして主流となっている物には, TCP(Transmission Control Protocol) と UDP(User Datagram Protocol) がある. Wind がこれらの既存の 2 つのプロトコルと違う点は 2 つある. 1 つは, 通信の同期・非同期といったメッセージタイプを, ユーザが send_to() などのシステムコールのフラグによって指定することができる点である. もう 1 つは, S-DSM と Wind の応用である. 以下, この 2 点について述べる.

1 点目の通信のメッセージタイプについて述べる. Wind は, 3 種類のメッセージタイプを指定することが可能である. 3 種類のメッセージタイプは, 以下の性質を持つ.

- 非同期通信であり到達順序と到達保証の両方を保証する
- 非同期通信であり到達保証のみを保証する
- 同期通信であり到達順序と到達保証の両方を保証する

順序を保証せずに到達保証のみを行うメッセージについて述べる. 例えば, マシン A からマシン B までのメッセージ転送を考える. A から B までのメッセージ転送において, 順序が必要とされるのは, 前後のメッセージにそれぞれ依存性がある場合である. しかし, 例えば, 10M バイトのメッセージを転送する時に, 最初の 4k バイトのパケットと最後の 4k バイトのパケットの到着順序が逆になったとしてもメッセージ全体のすべての到着が保証されれば, プログラムの実行には問題ない. この様な場合に, 到達順序の保証しないことで, より高いスループットが期待できる.

次に, 2 点目の S-DSM と Wind の応用について述べる. S-DSM システムは, Wind に対して 5 つの処理を指定できる. ここで, S-DSM システムにおける一般的なデータ受信処理を考える. まず, 受信したデータは, recv_from() などのシステムコールによって, カーネル内のソケットに使用されているバッファからユーザプロセスの受信用バッファに格納される. その後, S-DSM システムにより, 共有変数に対応する適切な仮想アドレスにコピーされる. この過程から, メッセージを受信した時とデータを S-DSM の共有メモリにコピーする時の 2 度のコピーが発生していることが分かる. その際, 2 度のシステムコール呼び出しと, 2 度のプロセス間コンテキストスイッチが発生する. Wind を使用することによって, このシステムコール呼び出しとプロセス間コンテキストスイッチを削減することができる. この処理により, S-DSM のパフォーマンスの向上が見込める.

現在, Wind は Linux[3][4][5][6] 上で設計及び実装を行っている. 実装を行うにあたり Linux カーネルバージョン 2.4.9 をベースに開発を進めている.

本稿では, 第 2 章で Wind の通信メッセージタイプと S-DSM への Wind の応用の設計について詳しく述べる. 第 3 章では, Wind の Linux カーネル上への

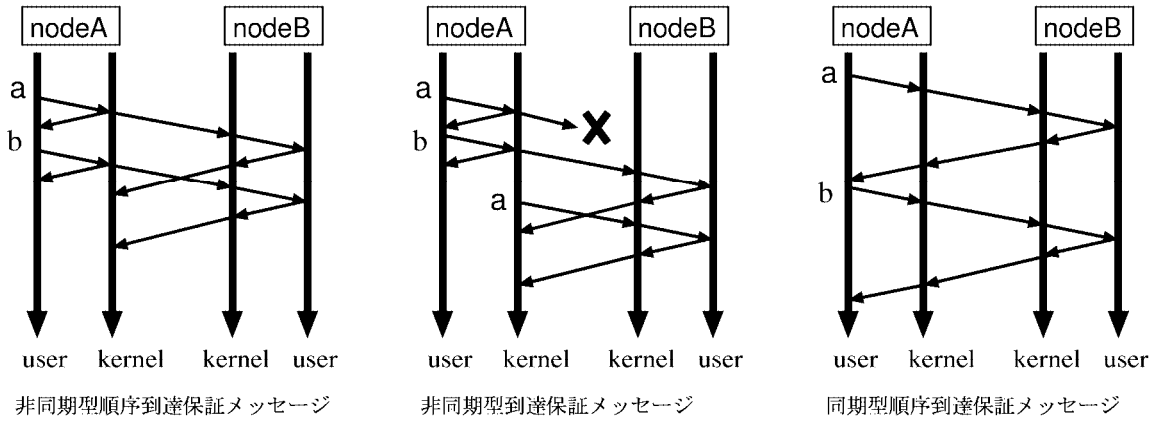

図 1: メッセージタイプ

実装について述べる．第 4 章では, Wind を用いた実測結果を記し, それに対する考察を行う．そして, 最後に第 5 章でまとめを述べる．

2 Wind の設計

本章では，Wind の特徴である Wind の通信メッセージタイプと S-DSM への Wind の応用についての設計方針について述べる．

2.1 通信メッセージタイプ

本節では，Wind が扱う 3 種類のメッセージタイプについて述べる．図 1 に, 3 種類のメッセージタイプのメッセージの流れ図を示す. 図 1 では, 隣り合った下方向の矢印 2 本で一台のマシンを表している．USER と KERNEL は, それぞれプロセスの実行モードを表している．以下, 3 種類のメッセージタイプについて説明する．

非同期型順序到達保証メッセージ

メッセージの到達と順序を保証する．ユーザプログラムからは, 送信要求だけ受け付け, 直ちに処理をユーザプログラムに返す.

非同期型到達保証メッセージ

メッセージの到達のみを保証をする．ユーザプログラムからは, 送信要求だけを受け付け, 直ちに処理をユーザプログラムに返す.

同期型順序到達保証メッセージ

メッセージの到達と順序を保証をする．ユーザプログラムからは, 送信相手からのメッセージの確認応答を受信してから, 結果と共に処理をユーザプログラムに返す.

Wind では, 同期型メッセージと非同期型メッセージをユーザが任意に指定することができる．このために, 異なるメッセージタイプ間での通信順序を規定する必要がある．ここで, 非同期型到達保証メッセージを送信した後に, 同期型順序到達保証メッセージを送信した時を考える．順序を保証するため, 先に送信されている非同期型到達保証メッセージがすべて到達するのを確認してから同期型順序到達保証メッセージを送信する．こうすることで, 順序保証が必要なメッセージは, 異なるメッセージタイプとの順序も保証している．

2.2 S-DSM への応用

S-DSM との応用では，Wind は，Wind 用ヘッダ (第 3.1.2 項参照) に S-DSM 用フラグを用意し，データの処理方式を切り替えている．ユーザプログラムが S-DSM 用フラグを設定するには, send_to() などの引数 (フラグ) に指定する．本章では，Wind の S-DSM 用フラグとその処理について述べる．

Wind には 5 種類の S-DSM メッセージタイプがある．これらの S-DSM メッセージタイプの詳細について以下に述べる．なお，Direct Diff Decode と

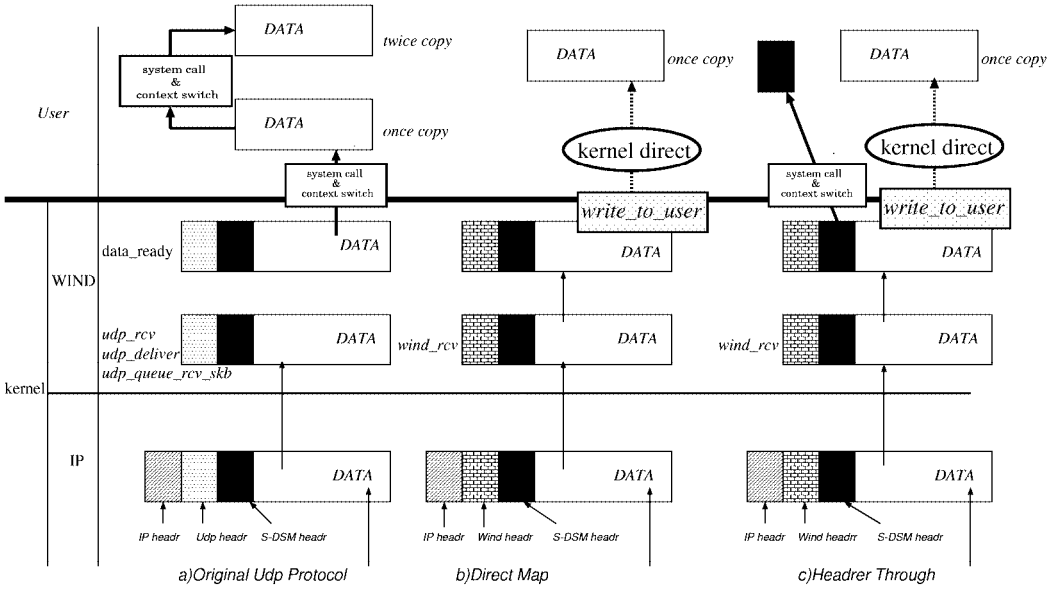

図 2: Wind と S-DSM の応用

Header Diff Decode は現段階では未実装である.

Direct Map Wind は, 受信した Wind のバッファ内に置かれているデータを, 直接受信プロセスの仮想メモリ空間にマッピングする. その後, Wind 用のヘッダは破棄する. このため, Direct Map 方式では, 受信プロセスは, ページを受信したことを知ることができない.

Header Through 基本的な動作は Direct Map 方式と同じである. 唯一の違いは, S-DSM 用のヘッダがカーネルによって破棄されることなく, ヘッダのみがユーザプロセスに渡されることである. これによって, ユーザプロセスはパケットの受信を知ることができる.

Message Through 既存の Wind プロトコルと同等の処理を行う. この方式では, データとヘッダの両方ともユーザプロセスへ渡される.

Direct Diff Decode このメッセージタイプは, S-DSM を特に意識した方式である. Direct Diff Decede は, Direct Map がページ単位で転送を行っていたのに対して, ページの更新された部分のみを転送するという特徴をもつ. 他の方法は, Direct Map と同じである.

Header Through Diff Decode Direct Diff Decode と同様に, ネットワークの転送単位はページではなくページの更新された場所のみである. Direct Diff Decode との違いは, ヘッダをユーザプログラムに渡すという点である.

3 Wind の実装

本章では, Linux カーネルバージョン 2.4.9 上への Wind の詳細について述べる.

3.1 データ構造

本節では, Wind におけるカーネル内データ構造について述べる.

3.1.1 カーネル内ソケット構造体

Wind を用いて通信を行う場合, ソケット作成時に SOCK_WIND をソケット作成システムコール (socket()) の引数として渡す. これによって, このソケットに対する送信・受信処理は Wind が処理を行う.

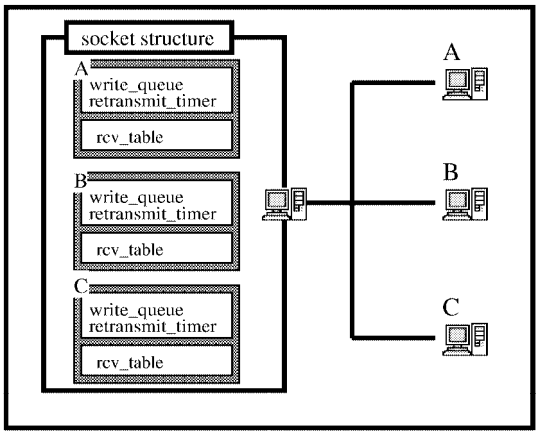

図 3: ソケットリソース

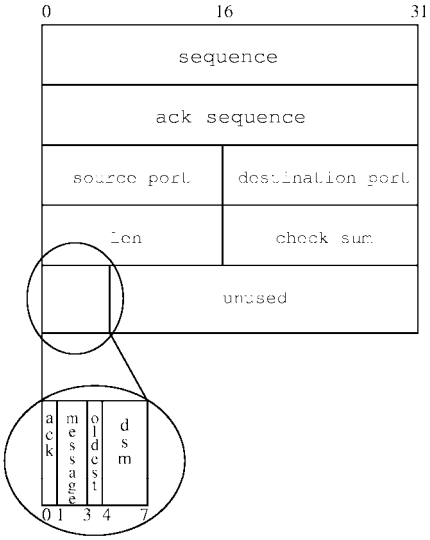

図 4: Wind ヘッダ構造

Wind の通信は, データグラム通信に分類できる. コネクション指向型通信は, 1 ソケットに対して通信相手が 1 台である. これに対し, データグラム通信は, 1 ソケットに対して複数の台数の端末と通信する. UDP では, 到達を保証しないために通信相手毎のソケット管理は不要である. 一方, Wind では, 到達保証を行うために 1 つのソケットで, 複数台の通信相手の管理を可能にしている. カーネルへの大きな変更点として, カーネル内部の socket 構造体に, 通信相手毎の送信キュー (図 3 の write_queue) と再送タイマ (同 retransmit_timer) と受信パケットテーブル (同 rcv_table) を追加した. 受信パケットテーブルは, 非同期型到達保証メッセージの受信時に使用する. これは, 非同期型到達保証メッセージは, ユニークなメッセージが届いたことが重要であり, 順序を必要としない. それゆえ, 重複したメッセージが届かないようパケット到着時に, シーケンス番号を逐一記憶する必要がある. その記憶のために受信パケットテーブルを使用する. 受信パケットテーブルは固定サイズで有限であるため, メッセージが多数送られてきた場合には, テーブルは再使用する必要がある (第 3.1.2 項参照). 受信パケットテーブルは, 通信相手毎に必要なため, テーブルが大きくなりすぎると無駄なメモリを使うことになる. 逆に, 小さすぎると何度もテーブルを初期化する必要がある. 現在の実装では, 受信パケットテーブルは相手毎に 4k バイトのサイズの配列を用いている.

3.1.2 Wind ヘッダ

本項では Wind のヘッダ構造について述べる. 図 4 に Wind ヘッダ構造を示す. ヘッダに含まれる内容は, シーケンス番号 (図 4 の sequence), 確認応答用シーケンス番号 (同 ack sequence), 送信元 (同 source port), 先 (同 destination port) ポート番号, データ長 (同 len), チェックサム (同 check sum), そして, 各種情報を表すビット列 (図 4 拡大部) で構成されている. 各種情報は, 次の情報を示す. message フィールドは, Wind メッセージタイプを表す. oldest フィールドは, パケット送信時に送信キューの先頭に繋がっている一番古いシーケンス番号を, ack sequence フィールドに値を入れているかを表す. これは, 限られた大きさの受信パケットテーブルの初期化を行うために, 最も古いパケットのシーケンス番号が必要となるためである. Wind の送信キューでは, 送信キューの先頭のパケットが一番古いパケットである. ack sequence フィールドは確認応答メッセージ外のメッセージの場合は使用しない. ゆえに oldest フィールドがセットされている場合のみ, 送信キューの中で最も古いパケットのシーケンス番号を ack sequence フィールドに格納できる.

3.2 到着保証

到着保証を行うために Wind では, 1 つのパケットが届くと 1 つの確認応答メッセージを返す. TCP では, 複数の確認応答メッセージをまとめて返す. Wind ではこれを行っていない. S-DSM の通信では, できるだけ送信元に確認応答を返し, 処理を S-DSM システムに返すのが重要なためである. パケットの再受信を防ぐためには, 受信パケットテーブルを用いる. 再受信が発生した場合, 再受信パケットは無視され, 確認応答メッセージを送信元に送信する.

3.3 順序保証

順序保証を行うメッセージの順序保証について述べる. 順序保証は, シーケンス番号を用いて行う. 受信側で期待したシーケンス番号以外のパケットが送信された場合には, 受信を拒否する. パケットの受信を失敗すると, それ以降のシーケンス番号のパケットは, 受信を失敗したパケットが到達するまで受信を拒否する. 再送タイマによって, 受信に失敗したパケットが到着すると, それ以降のパケットの受信も行われる. Wind では. このように順序保証を保証している.

3.4 S-DSM 用データ受信処理

メッセージを受信した場合, パケットはイーサネット又はギガビットイーサネットを通り, IP 層に至る. その後, Wind に処理が移る. Wind は, 受信パケットテーブルを参照し, 到着パケットが重複して受け取っていないかを調べる. 次に, message フィールドを参照し, どのメッセージタイプかを判断し, 受信処理を進める. その後, dsm フィールドを参照し, S-DSM に対し, 指定されている S-DSM メッセージタイプに対応した処理を行う. Wind は, S-DSM 用の処理を行う場合, S-DSM が使用する S-DSM ヘッダを参照する. S-DSM はその性質上, 一度ユーザプロセスが受信したデータを同じユーザプロセス上の異なった仮想アドレスへとコピーする (図 2 参照). この同一プロセス間の仮想アドレスへのコピーを, Wind ではカーネルコンテキストで行う. カーネルコンテキスト上で S-DSM が扱う仮想メモリを扱うためには, コピー対象となる仮想アドレスをカーネルコンテキスト上で扱えるように実メモリに変換する必要がある. これには, i386 系プロセッサの cr3 レジスタを用いる. cr3 レジスタはプロセスに固有な値である (図 5 参照). cr3 レジスタを基に仮想アドレスを用いて実アドレスに変換する. Wind は, 変換後の実アドレスに, データをコピーする. この後, Direct Map 方式では, 受信パケットを解放し, ユーザプロセスに対して何の通信も行わない. 一方, Header Through 方式では, S-DSM ヘッダのみを S-DSM に渡す. このことによって, ユーザプロセスは受信パケットの到着を知ることができる (図 2.2 参照).

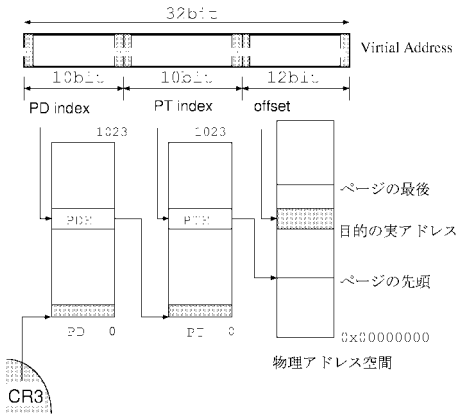

図 5: 仮想アドレスから実アドレスへの変換

表 1: 実験環境

CPU	Pentium III 800MHz
Main Memory	512Mbyte
OS	Linux Kernel 2.4.9 (Wind 用に変更)
Network	1000Mbps Gigabit Ethernet with Switching Hub
Network Card	Intel PRO/1000F ServerAdapter

4 評価

本章では, Wind を用いた評価の結果を述べている. なお, 評価を行う環境は表 1 の通りである. 評価内容には, スループットと S-DSM への応用 (バリア同期) についての評価を行った.

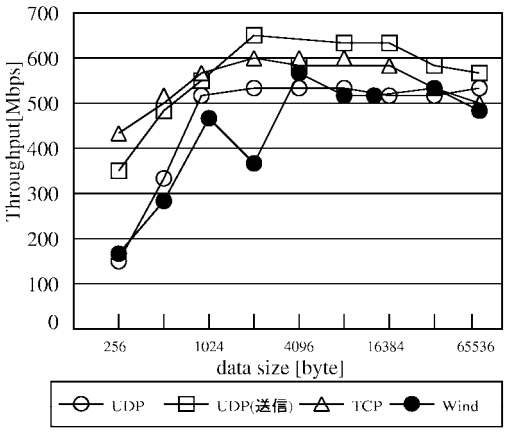

図 6: 通信スループット

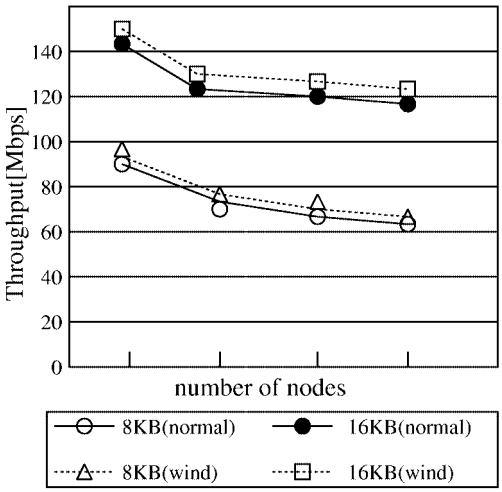

図 7: N ノード間でのバリア同期

4.1 スループット

Wind の通信スループットの計測を行った. 比較評価のため, TCP と UDP の通信スループットについても計測を行った. 図 6 に結果を示す. なお, Wind の測定結果は非同期到保証メッセージで行った. 図 6 の UDP(送信) は, UDP の送信関数の返り値である. UDP のグラフは, UDP を用いて受信した場合に, 可能な限り受信した場合の値である. UDP(送信) と UDP の差は, 測定時間内にパケットの受信を失敗した量をあらわしている.

4.2 S-DSM (Fagus) との応用

本節では, S-DSM システムの Fagus[7][8] に, 第 2.2 節で述べた Wind の S-DSM との応用部分のみを適応した時の評価を行った. 図 7 に Fagus を Wind 上に実装した場合のバリア同期のスループットを示す. 次に, Fagus のバリア同期における処理について述べる. ユーザプログラム中のバリア同期点に達したノードは, 更新された共有メモリの一貫性を保証するために, 共有変数のオーナーにバリア同期点に到達したことを通知する. オーナーは, バリア同期点にすべてのノードが到達したことを確認する. その後, オーナーは, 自分以外のノードに対し, 共有変数に対応付けられた共有メモリを送信する. これは, 結果として, オーナー以外のノードに対し, 一斉に更新されたデータを送信することになり, S-DSM システムにおいて高速化が求められる処理である. 図 7 のサイズは, バリア同期によって, データを更新する必要がある共有メモリの大きさを示す.

4.3 考察

第 4.1 節では, 全体的に TCP が高い数値を表している. これは, TCP がパケットサイズを MTU (Maximum transmission unit) に近づけてから送るためである. 特に小さいパケットを連続して送る時は, 複数のパケットを 1 つにまとめることにより, 通常複数回必要な送信処理を 1 度で行うことが出来る. この手法は, 全体的なスループットを確保するのに非常に有効であると考えられる. また, TCP では, 確認応答メッセージを一つにまとめて送るという手法により, 送信側が受ける確認応答のパケット数を減少させている. その結果, より送信効率を上げることができていると考えられる. Wind と UDP の比較をした場合, ほぼ同等の通信性能であることが分かる. このことから, UDP を用い, 通信の保証が必要なアプリケーションでは Wind が有効であると考えられる.

ここで, Fagus の通信特性について見てみる. 表 2 は Fagus の通信特性を表している. この表は, アプリケーション毎に Fagus が行う通信のパケットを表している. Fagus は UDP による通信を行う. 256 バイト以下の小さなパケット群は, その多くが S-DSM を制御するためのコントロールメッセージである. これらのメッセージは, 通信スループットよりもラウン

表 2: Fagus の通信特性 [単位：パケット個数]

パケットサイズ (byte)	256	512	1024	2048	4096	8192	16384	32768	65536
LU 1024x1024x8	901	0	0	0	0	128	0	0	0
LU 2048x2048x8	1797	0	0	0	0	0	256	0	0
LU 4096x4096x8	3589	0	0	0	0	0	0	512	0
himenoBMT[9]	1605	0	0	0	0	0	400	0	0

ドトリップタイムが重要になる．実際に共有メモリ転送を行う場合では, 8k バイト以上のメッセージを使うことが多い．Wind では, S-DSM の通信で必要となるラウンドトリップタイムの反応を早くするために 1 パケットにつき, 1ack を返している．このことは, Wind のベンチマークでの通信性能が悪くなっている一因だと考えられる．

一方, 第 4.2 節では, すべての台数において, Wind を適応することで, バリア同期時のスループットが向上することが分かった．これは, S-DSM に Wind を適応することによって S-DSM のパフォーマンスの向上が期待できることを意味している．

5 おわりに

我々は, S-DSM に適した通信プロトコル Wind の設計と実装を行った．評価において, 既存プロトコルの TCP と Wind で大きくスループットに開きが出る結果となった．この評価を受けて, Wind においても, TCP が用いている送信パケットサイズを MTU にできるだけ近づけて送信する方法を実装する必要があると考える．また, 現在はマルチキャストのサポートは行っていない．しかし, マルチキャストを用いることにより, 送信回数を減らすことができる．これは,

結果として, S-DSM の性能を向上させることができる．今後, リライアブルマルチキャストの研究を基に実装を行う予定である. 評価項目においては, 今回は行うことが出来なかった．Wind の 2 つの特徴 (S-DSM に対応した通信処理とメモリアクセス) を合わせた評価を行う予定である．S-DSM との応用では, S-DSM のバリア同期のスループットを上げることに成功した．今後も, S-DSM を高速化するという方針での Wind の開発を行っていく予定である．

参考文献

[1] Shoichi Saito,Akihito Hayashi,Tetsutaro Uehara,Kazuki Joe and Yoshitoshi Kunieda,a Low-cost Comminication Module for Software DSM Systems,In Proceeding of the International Conference on PDPTA '2000, pp. 721-727 (2000).

[2] 林 章仁, 齋藤彰一, 上原哲太郎, 國枝義敏:
UDP を用いた高速通信ライブラリ Wind の実装,
情報処理学会研究会報告 2000–OS–85, Vol. 2000, No. 75, pp. 9–16 (2000.8).

[3] R. Card,E. Dumas and F. Mevel,Linux2.0 カーネルブック, オーム社出版局 (1999).

[4] Michel Beck,Harald Bohme,Mirko Dzidzka,Robert Magnus, Ulrich Kunitzand and Dirk Verwoner,Linux カーネルインターナル, 株式会社アスキー (1999).

[5] ScottMaxwell,Linux Core Kernel, 小学館 (2000).

[6] Daniel P. Bovet,Marco Cesati, 詳解 LINUX カーネル, オライリー・ジャパン (2001).

[7] 横手聡, 齋藤彰一, 上原哲太郎, 國枝義敏:コンパイラによる制御が可能な DSM システム「Fagus」の実現, 情報処理学会研究会報告 2000–OS–85, Vol. 2000, No. 75, pp. 47–54 (2000.8).

[8] 横手聡, 林 章仁, 齋藤彰一, 上原哲太郎, 國枝義敏:高速通信ライブラリ Wind を用いたソフトウェア分散共有メモリシステム Fagus の性能評価, 情報処理学会研究会報告 2001–OS–88, Vol. 2001, No. 78, Vol. 2001, (2001.7)

[9] 姫野 龍太郎,Himeno benchmark xp – 姫野ベンチとは,http://w3cic.riken.go.jp/HPC/HimenoBMT/index.html