Vol.2016-OS-137 No.13
2016/5/31

MPI通信パターンに基づくSDN制御を高速化するカーネルモジュールの試作と評価

高橋 慧智[1,a)]　伊達 進[2,b)]　Khureltulga Dashdavaa[1,c)]　木戸 善之[2,d)]　下條 真司[2,e)]

概要：われわれは，ネットワーク内のパケットフローを動的に制御可能とするSDNに着眼し，並列分散プロセス間通信ライブラリMPIの通信性能向上を目的とし，SDN-MPIの研究開発を推進してきた．SDN-MPIのプロトタイプ実装を通じて，本研究では，個別のMPI関数を単独で実行した際の通信時間の短縮を実証した．しかし，これまでの実装は，MPIの個別の集団通信の高速化を実現するに留まっており，複数の集団通信が駆使された実際のMPIアプリケーションへの応用のためには，アプリケーションが呼び出すMPI関数相互結合網の制御を連動・連携して行う仕組みが必要不可欠である．本稿では、そのような問題点に着眼し、われわれが開発してきたMPI通信パターンに基づくSDN制御を高速化するカーネルモジュールについて報告する．

キーワード：Message Passing Interface, Software-Defined Networking, Loadable Kernel Module

1. はじめに

近年，HPC（High Performance Computing）システムを構成する計算ノード数の増加が顕著である．HPCシステムにおける計算ノードは，相互結合網と呼ばれる高性能ネットワークにより互いに接続されている．今日の相互結合網は，任意の計算ノード間で可能な限り有効帯域幅を最大化，かつ遅延を最小化するように，十分な余裕を持って設計される．しかし，このような十分に余裕を持たせた設計方針では，ノード数のスケールアウトに伴い，相互結合網のトポロジは大規模化かつ複雑化することが予測される．大規模かつ複雑なトポロジを有する相互結合網は，それを構成するために必要となるスイッチやリンクの数が膨大となるため，構築コストの観点から今後ますます実現が困難になっていくものと考えられる．

そのような背景から，われわれは，実際のネットワーク構成およびトラフィック状況等に応じて相互結合網内で発生するノード間通信を動的に制御することで，HPCシステムの相互結合網の利用率を高めることを目的とした研究開発を推進してきた．技術的には，大規模HPCシステム上で動作する並列分散アプリケーションの通信性能を向上させることを目的とし，ネットワーク内のパケットフローを動的に制御可能とするSDNを，並列分散プロセス間通信ライブラリMPIに応用したSDN-MPIを提案・実装してきた [1,8]．`MPI_Bcast` [1] では，MPIの集団通信の一つである`MPI_Bcast`をターゲットとし，HPCシステムの相互結合網のトポロジおよび実行時のプロセス配置情報をもとに，最適なブロードキャスト配信を行う手法を実証した．また，`MPI_Allreduce` [8] では，冗長経路を含むトポロジに着目し，相互結合網内のトラフィックに応じてリンクの負荷分散を行うことで，リンクにおける輻輳を回避する手法を提案・実装した．

これらの手法・実装は，MPIの個別の集団通信の高速化を実現するに留まっており，複数の集団通信が駆使された実際のMPIアプリケーションへの応用のためには，アプリケーションが呼び出すMPI関数と相互結合網の制御を連動・連携して行う仕組みが必要不可欠である．本稿では，そのような問題点に着眼し，われわれが開発してきたMPI通信パターンに基づくSDN制御を高速化するカーネルモジュールについて報告する．

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
[2] 大阪大学大学サイバーメディアセンター
Cybermedia Center, Osaka University
[a)] takahashi.keichi@ais.cmc.osaka-u.ac.jp
[b)] date@cmc.osaka-u.ac.jp
[c)] huchka@ais.cmc.osaka-u.ac.jp
[d)] kido@cmc.osaka-u.ac.jp
[e)] shimojo@cmc.osaka-u.ac.jp

2. 技術課題

本節では，われわれが提案する SDN-MPI を概説し，本稿で取り扱う技術課題について記す．そのために，まず SDN-MPI の構成技術である SDN（Software-Defined Networking）と MPI（Message Passing Interface）について説明する．

2.1 SDN（Software-Defined Networking）

SDN（Software-Defined Networking）[7] はパケットの流れを動的に集中制御可能にするネットワークアーキテクチャである．従来のネットワークアーキテクチャでは，パケット制御の決定を担うコントロールプレーンと実際のパケット転送を担うデータプレーンは，単一のネットワーク機器上に一体の機能となって実装されていた．SDN では，図 1 に示すように，コントロールプレーンとデータプレーンをそれぞれ別々の機器に分離する．SDN スイッチと呼ばれるネットワーク機器はデータプレーンの処理，SDN コントローラはネットワークのコントロールプレーンの処理をソフトウェアにより担当する．この機能分離により，ネットワークの動的なプログラミング制御が実現される．

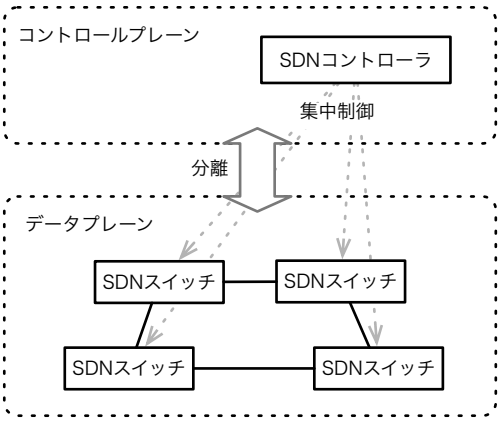

図 1 SDN のアーキテクチャ

SDN では，パケットの流れの制御を「フロー」という単位で行う．各 SDN スイッチは，フローの集合である「フローテーブル」を保持する．表 1 にスイッチが保持するフローテーブルの一例を示す．フローは，対象のパケットを限定するヘッダフィールドと，パケットに対して行う操作を定義するアクションから構成される．ヘッダフィールドの例として，TCP/UDP ポート番号，IP アドレス，MAC アドレスがある．一方，アクションには，パケットを特定のポートへ転送する，パケットのヘッダを書き換える，パケットをドロップするなどの操作を指定可能である．

SDN スイッチはパケットを受信すると，マッチするフローをフローテーブルから検索し，該当フローのアクションを実行する．SDN コントローラは，ネットワーク内の全ての SDN スイッチのフローテーブルを操作することで，ネットワーク内のパケットの流れを制御する．今日では，SDN の一実装として，OpenFlow [4] が業界標準規格となりつつある．

表 1 フローテーブルの一例

ヘッダフィールド				アクション
宛先 MAC	送信元 IP	宛先 IP	···	
	192.0.2.12	192.0.2.34		ポート 1
	192.0.2.34	192.0.2.56		ポート 2
ff:ff:ff:ff:ff:ff				ポート 1,2
72:42:c1:e4:75:8c				ドロップ

2.2 MPI（Message Passing Interface）

MPI（Message Passing Interface）[3,5] は，メッセージパッシング方式に基く，並列分散アプリケーションの開発に用いられるプロセス間通信ライブラリである．メッセージパッシング方式では，並列分散プロセス同士は，メッセージと呼ばれるデータの交換により協調して動作する．MPI は，相互結合網を構成するハードウェアや通信プロトコルの差異を吸収するため，アプリケーションの開発効率と可搬性が高い．そのような理由から，HPC システムで動作する並列分散アプリケーションの多くが MPI ライブラリを用いて実現されている．そのため，MPI の通信性能の改善は，アプリケーションの実行時間の短縮へとつながる．

MPI によるメッセージ通信は，1 対 1 通信と集団通信に大別される．1 対 1 通信関数は，2 つのプロセス間の通信を取り扱う．`MPI_Send` や `MPI_Recv` はその代表例である．一方，集団通信関数は複数のプロセスから構成されるグループ内の通信を取り扱う関数である．あるプロセスから他の複数のプロセスに同一のデータを配信する `MPI_Bcast` や，複数のプロセスから 1 つのプロセスにデータを縮約する `MPI_Reduce` は集団通信関数の一例である．

これらの MPI 通信関数は引数として，通信に参加するプロセスを指定する．1 対 1 通信の場合は送信元プロセス・宛先プロセスの識別子，集団通信の場合は通信に参加するプロセスのグループの識別子を，それぞれ引数として渡す．本稿では，以下，MPI 関数の種類，これらの参加プロセスの情報を合わせて，通信パターンと定義する．

2.3 SDN-MPI 実現のための技術課題

われわれは，2.1 節で述べた SDN によるフローの動的制御機能に着眼し，並列分散プロセス間通信ライブラリ MPI の通信性能向上を目的とし，SDN-MPI の研究開発を推進してきた．図 2 に提案する SDN-MPI のコンセプトを示す．SDN-MPI では，MPI アプリケーションから通信パターンを抽出し，この抽出された通信パターンから通信性能を向上するために最適なフロー制御を実現する．この

コンセプトに基づき，われわれは MPI の集合通信である MPI_Bcast，MPI_Allreduce の通信パターンを抽出し，その通信性能を向上させる研究開発を推進してきた．その結果，通信パターンに応じて，相互結合網内を流れるネットワークフローを制御することにより，それぞれの集合通信の高速化が可能であることを示してきた [1, 8].

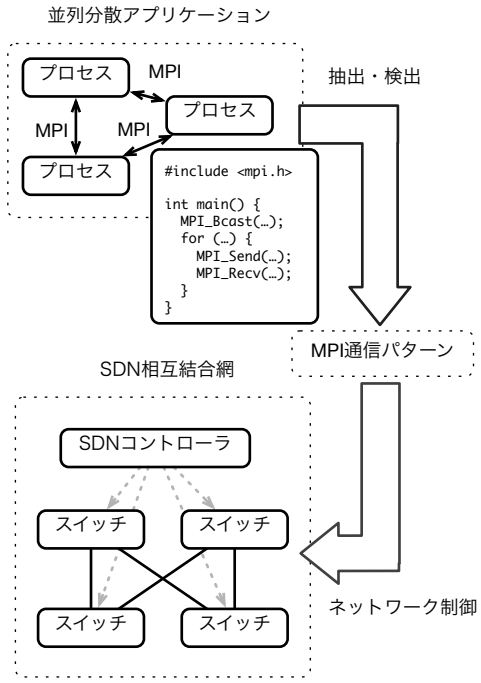

図 2 SDN-MPI 概念図

しかし，複数の MPI 関数を用いる実際の MPI アプリケーションの実行に合わせて発生する通信パターンに応じて，相互結合網内を流れるパケットフローを制御する仕組みは未だ実現できていない．実際の MPI アプリケーションでは，単一のアプリケーションが複数の MPI 関数を実行する．そのため，アプリケーションが MPI 関数を呼び出すごとに，その通信パターンをリアルタイムに把握する必要がある．さらに，通信パターンに応じて，通信性能を向上するように相互結合網内のフローの制御を行わなければならない．本研究では，MPI アプリケーションの実行に応じて，ネットワーク内のパケットフローを制御可能にする仕組みを実現するに際して，以下を技術課題として設定する．

1. オーバーヘッドの削減：アプリケーションの実行とフローの制御は，可能な限り小さいオーバーヘッドで同期しなければいけない．
2. スケーラビリティの向上：SDN スイッチにインストールできるフロー数の上限はできるだけ高く設定できなければならない．
3. MPI に由来するパケットの識別：相互結合網内には SSH など，MPI と無関係なパケットも存在する．そのため，MPI が送出したパケットのみを判別し，制御する必要がある．

3. 提案

3.1 提案手法の概要

提案手法の基本的な考え方は，MPI が送出する各パケットに通信パターンをエンコードしたタグを付与し，タグに基いてネットワークでパケット制御することにある．図 3 に提案手法の概要を示す．各計算ノードは，ユーザ空間で 1 つ以上の MPI アプリケーションのプロセスを実行する．MPI アプリケーションは，MPI ライブラリを動的にリンクしている．計算ノードのカーネル空間には，「タグ付けカーネルモジュール」が常駐し，MPI ライブラリが送出するパケットに，タグ付け処理を実行する．MPI ライブラリとカーネルモジュールは連携動作し，タグ付けに必要な MPI ライブラリ内部の情報を共有する．

SDN コントローラは，MPI パケットのタグに基いて，パケットを制御するフローエントリを各スイッチのフローテーブルに登録する．スイッチは MPI パケットを受信するとパケットのタグを読み取り，フローテーブルにしたがってパケットを処理する．受信側の計算ノードに隣接するスイッチは，計算ノードにパケットを転送する前に，タグを除去する．

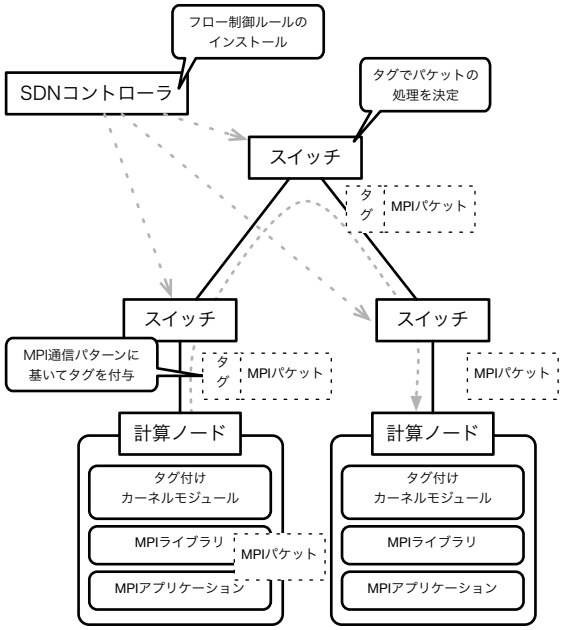

図 3 提案手法の概要

タグ付けカーネルモジュールは，MPI というアプリケーションのレイヤの情報を SDN で取り扱えるように，タグという形でパケットに埋め込む役割を担っている．パケット自身に MPI 通信パターンが書き込まれているため，SDN コントローラやスイッチは，MPI アプリケーションと別途通信することなく，各々のパケットに対する制御を決定できる．本提案手法により，低オーバーヘッドな MPI アプ

リケーションとネットワーク制御の同期が可能になる．

3.2 タグ付けカーネルモジュール

3.2.1 タグの格納場所

提案手法では，タグに基くパケット制御を効率的に実行できるようにするため，タグを MPI パケットの MAC アドレスフィールドに書き込む方法を選択した．この選択をした理由は 2 つあり，SDN スイッチにおけるパケット転送処理のオーバーヘッドの削減と，フロー数のスケーラビリティの向上である．

OpenFlow スイッチが取り扱えるヘッダフィールドは，OpenFlow 規格で定義されたフィールドに限られる．事前定義されていないヘッダフィールドを読み取る場合は，毎回 SDN コントローラへパケットを転送する必要があり，オーバーヘッドが非常に大きい．そのため，タグは，OpenFlow 規格で定義されていヘッダフィールドのいずれかに埋め込むのが望ましい．MAC アドレスフィールドは，OpenFlow 規格において定義されているヘッダフィールドである．

さらに，スイッチに登録できるフローの数が増えるという利点がある．OpenFlow スイッチのフローテーブルに登録できるフローの数には上限があり，一般的な OpenFlow スイッチでは数千個のオーダである．しかし，多くの OpenFlow スイッチは L2 ヘッダフィールド（MAC アドレスと VLAN ID）のみ含むフローに関しては数万個登録できる．それゆえ，タグをパケットを MAC アドレスに格納することで，他のヘッダフィールドに格納するよりも，フロー数の上限を増やせる．

3.2.2 MAC アドレスの書き換え

パケットのアドレスフィールドを動的に変更する方法として，Raw ソケットを用いる方法，netfilter API を用いる方法，プロトコルハンドラを用いる方法などがあるが，提案手法ではプロトコルハンドラを用いる方法を選択した．以下にその理由を述べる．

Raw ソケットは，ユーザ空間のプログラムからパケットのプロトコルヘッダを読み書き可能にする特殊なソケットである．Raw ソケットを用いれば，L2 ヘッダを自由に構成してパケットを送信できる．一方で，ユーザ空間で TCP/IP スタックを全て再実装する必要があり，非現実的である．また，MPI ライブラリを Raw ソケットを使用するように大きく修正する必要があることや，アプリケーションが root 権限で実行されなければいけないなどの欠点もある．

Linux において，パケットのフィルタリングや改変のためには，Linux カーネルが提供する netfilter API が一般的に用いられる．実際，Linux におけるファイアーウォールや NAT は，内部で netfilter を使用している．Netfilter は，プログラムがカーネルのネットワークスタックの特定の場所にフックし，独自の処理を挿入することを可能にする．しかし，フックできる場所は事前定義された場所に限られ，カーネルがパケットに L2 ヘッダを付与するのは，netfilter でフック可能な場所を通過した後である．そのため，netfilter ではパケットの MAC アドレスを書き換えることはできない．

以上の議論を踏まえて，提案手法では，プロトコルハンドラを実装することで MAC アドレスの書き換えを実現している．プロトコルハンドラは，パケットが NIC（Network Interface Card）から到着した直後と，NIC へ送信される直前にカーネルが呼び出すコールバック関数である．プロトコルハンドラは，カーネル空間で実行されるため，カーネルモジュール [2] に実装する．プロトコルハンドラでは，NIC へ送信するパケット全体を取得し，書き換えることができるため，タグ付けを実装できる．

3.2.3 MPI パケットの識別

プロトコルハンドラは，カーネルのネットワークスタックが送出する全てのパケットについて呼び出される．つまり，MPI ライブラリ以外によって送信されたパケット，例えば計算ノードの制御に用いられる SSH のパケットについてもプロトコルハンドラが呼び出される．よって，タグ付け処理の前段階として，プロトコルハンドラに渡されたパケットが MPI ライブラリに由来するものか判定しなければいけない．提案手法では，あるパケットが MPI に由来するかの判定条件として，それぞれの MPI プロセスの組について一意に定まる，下記の 4 つ組を用いている：

- 送信元 IP アドレス
- 宛先 IP アドレス
- 送信元 TCP ポート番号
- 宛先 TCP ポート番号

以下では，この 4 つ組を「ピア情報」と呼ぶ．具体的な識別処理の手順は次の通りである．MPI プロセス間で TCP 接続が確立する度に，ピア情報をリストに追加していく．これを「ピアリスト」と呼ぶ．そして，プロトコルハンドラがパケットを受信すると，そのパケットからピア情報を抽出し，ピア情報がピアリストに含まれているか確認する．ピアリストに含まれていれば，パケットは MPI ライブラリによって送出されたパケットであるから，タグ付け処理に続行する．ピアリストに含まれていなければ，単純に無視する．

3.2.4 MPI ライブラリとの連係

前節で述べた MPI パケットの識別方法では，タグ付けカーネルモジュールが，MPI プロセス間で TCP 接続が確立したことをどのように検知するかという課題がある．提案手法では，これを MPI ライブラリとタグ付けカーネルモジュールの連係により実現する．具体的には，MPI ライブラリが MPI プロセス間で TCP 接続を確立した際に，タグ付けカーネルモジュールに前述のピア情報を通知する．本提案手法における各コンポーネントの相互作用を，図 4

に示す．なお，実線はコンポーネントの連係を，点線矢印はパケットの流れを表す．

ユーザ空間のプログラムとカーネルモジュールの間の通信には，procfs，sysfs，netlink ソケット，ioctl システムコールなど様々な実現方法がある．ここでは，転送するデータサイズが小さいことと，実装の容易さを考慮し，ここでは ioctl システムコールを採用した．ioctl は，ユーザプログラムからデバイスドライバを制御するためのシステムコールである．

具体的な連係手順は次の通りである．タグ付けカーネルモジュールは，misc キャラクタデバイスとして振る舞い，/dev/sdnmpi にデバイスファイルを公開する．MPI ライブラリは，初期化時に/dev/sdnmpi デバイスを開き，ファイルハンドルを保持しておく．そして，MPI プロセス間で TCP 接続が確立した際（connect あるいは accept 関数成功時）に，/dev/sdnmpi デバイスに対して ioctl システムコールを発行し，ピア情報を引数として渡す．カーネルモジュール側では，ioctl システムコールのハンドラでピア情報を受信し，ピアリストに追加する．

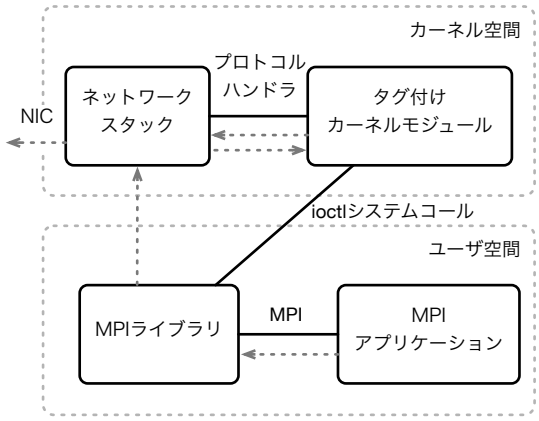

図 4 タグ付けカーネルモジュールと MPI ライブラリの連係

3.3 SDN コントローラ

本提案手法の SDN コントローラには，大きく分けて以下の 3 つの機能がある：

- MPI が送出するパケットのタグに基く制御
- MPI 以外のアプリケーションが送出するパケットの制御
- 相互結合網のトポロジの把握

MPI が送出するパケットは，3.2 節で説明したタグ付けカーネルモジュールにより，通信パターンを反映したタグが付与される．SDN コントローラは，通信性能を向上させるように，タグにマッチして何らかのアクションを実行するフローを SDN スイッチへフローをインストールする．通信パターンからフローを生成する具体的なアルゴリズムは，拡張可能な仕組みになっている．例えば，これまでにわれわれが開発した，SDN-MPI 版の MPI_Bcast や MPI_Allreduce [1,8] などが考えられる．

相互結合網内には，SSH や NFS など，MPI 以外のアプリケーションが送出するパケットも流れる．3.2.3 節で述べた通り，これらのパケットにはタグが付与されない．そのため，MPI が送出するパケットとは別に扱う必要がある．まず，パケットの送信元 MAC アドレスと宛先 MAC アドレスから，送信元計算ノードと宛先計算ノード間の経路を計算する．この際，複数の経路が可能な場合は，リンクの負荷が分散されるように，1 つ経路を選択する．最後に，経路に沿ってパケットが転送されるように，選択した経路に含まれる SDN スイッチに，フローをインストールする．

上記の 2 つの機能を実現するためには，SDN コントローラが相互結合網のトポロジを正しく把握する必要がある．提案手法では，LLDP（Link Layer Discovery Protocol）を用いてトポロジの検出を実現している．各 SDN スイッチに，自身の情報を含んだ LLDP パケットを定期的に送出させる．SDN スイッチが LLDP パケットを受信すると，LLDP パケットを解析し，隣接するスイッチに関する情報を取得する．この手順を全スイッチについて行うことで，SDN スイッチの隣接リストを構築する．

4. 評価実験

提案手法の有効性を評価するために，タグ付けによるオーバーヘッドの大きさを評価する実験を行った．マイクロベンチマークを利用して，2 ノード間での 1 対 1 通信の有効帯域幅と遅延を，提案手法を使用する場合としない場合で計測し比較した．

4.1 評価環境

評価には，SDN スイッチ 1 台に接続した 2 台の計算ノードを使用した．表 2 に SDN スイッチの性能を，表 3 に計算ノードの性能を示す．マイクロベンチマークには OSU Micro Benchmark 5.1 [6] を使用した．

表 2 SDN スイッチの性能

型番	NEC ® UNIVERGE PF5240
SDN 規格	OpenFlow 1.0
Ports	Gigabit Ethernet ×48
スイッチング容量	176Gbps
転送性能	131Mpps

表 3 計算ノードの性能

CPU	Intel ® Xeon ® CPU E5520 × 2
メモリ	12GB (ECC)
ネットワーク	Gigabit Ethernet
OS	Fedora 20 (Linux Kernel 3.19.8-100)

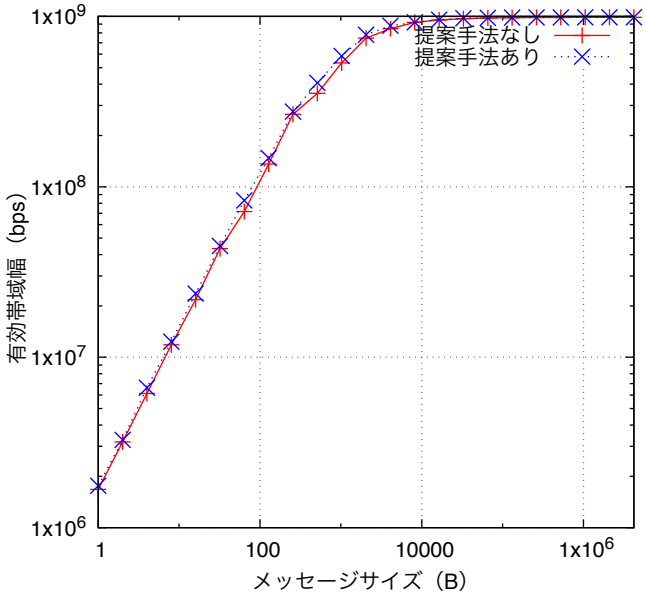

図 5 2 ノード間の有効帯域幅の比較

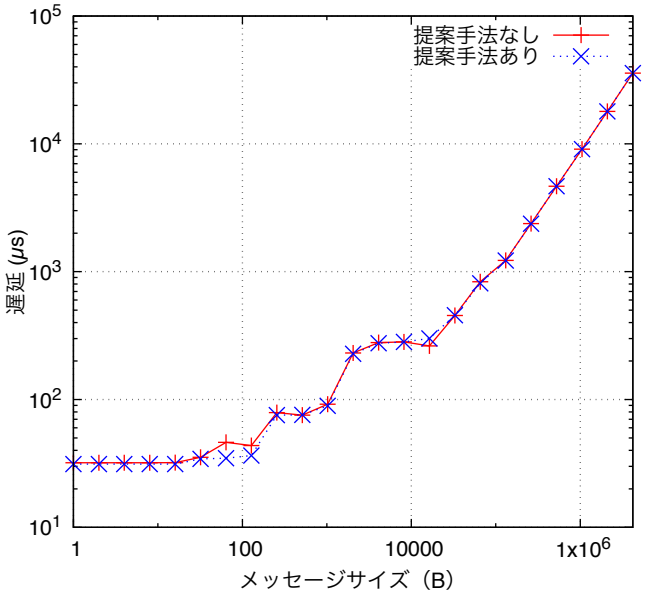

図 6 2 ノード間の遅延の比較

有効帯域幅の計測には OSU Micro Benchmark に含まれる `osu_bw` ベンチマークを使用し，ウィンドウサイズは 64，繰り返し回数は 500 回とした．一方，遅延の計測には `osu_latency` ベンチマークを使用し，繰り返し回数は 50,000 回とした．有効帯域幅と遅延の計測の両方で，送信するメッセージサイズを 1 バイトから 2M バイトまで変化させた．

4.2 評価結果

有効帯域幅のベンチマーク結果を図 5 に，遅延のベンチマーク結果を図 6 に示す．これら 2 つプロットにおいて，全てのメッセージサイズで提案手法を使用する場合としない場合で，大きな差が見られない．よって，提案手法のタグ付けによるオーバーヘッドは実用上無視できる程度であると言える．

5. おわりに

本稿では，SDN-MPI を複数の集団通信が駆使された実際の MPI アプリケーションへの応用するために，アプリケーションが呼び出す MPI 関数相互結合網の制御を連動・連携して行う仕組みを提案した．提案手法の基本的な考え方は，MPI が送出する各パケットに通信パターンをエンコードしたタグをカーネルモジュールによって付与し，タグに基いて SDN 相互結合網でパケット制御することにある．実機の計算ノードと SDN スイッチを用い，2 ノード間の有効帯域幅と遅延について提案手法のオーバーヘッドを評価した．評価実験の結果，提案手法によるオーバーヘッドは実用上無視できる程小さいことを実証できた．今後の課題として，実際のアプリケーションでの性能評価や，これまで開発してきた SDN-MPI の関数の統合がある．

謝辞

本研究は JSPS 科研費 JP26330145 の助成を受けた．

参考文献

[1] Dashdavaa, K., Date, S., Yamanaka, H., Kawai, E., Watashiba, Y., Ichikawa, K., Abe, H. and Shimojo, S.: Architecture of a High-speed MPI_Bcast Leveraging Software-Defined Network, *UCHPC2013: The 6th Workshop on UnConventional High Performance Computing* (2013).

[2] De Goyeneche, J.-M. and De Sousa, E.: Loadable Kernel Modules, *IEEE Software*, Vol. 16, No. 1, pp. 65–71 (online), DOI: 10.1109/52.744571 (1999).

[3] Gropp, W., Lusk, E. and Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-passing Interface, 第 1 巻，The MIT Press (1999).

[4] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J.: OpenFlow: Enabling Innovation in Campus Networks, *SIGCOMM Computer Communication Review*, Vol. 38, No. 2, pp. 69–74 (online), DOI: 10.1145/1355734.1355746 (2008).

[5] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard (2015).

[6] Ohio State University: OSU Micro Benchmark. `http://mvapich.cse.ohio-state.edu/benchmarks/`.

[7] Open Networking Foundation: Software-Defined Networking: The New Norm for Networks (2012).

[8] Takahashi, K., Khureltulga, D., Watashiba, Y., Kido, Y., Date, S. and Shimojo, S.: Performance Evaluation of SDN-enhanced MPI_Allreduce on a Cluster System with Fat-tree Interconnect, *International Conference on High Performance Computing & Simulation, HPCS 2014*, pp. 784–792 (online), DOI: 10.1109/HPCSim.2014.6903768 (2014).