適応スパニングツリーを用いた広域メッセージパッシングシステム用の集合通信

斎藤秀雄[†] 田浦健次朗[†] 近山隆[††]

本稿では，広域メッセージパッシングシステム用に，動的に生成したスパニングツリーを用いて集合通信を行う手法を提案する．提案手法では，効率良くブロードキャスト・リダクションを行うために，実行時に測定した遅延・バンド幅を基にスパニングツリーを構築する．プロセッサは，自律的に遅延とバンド幅を測定することによって，短いメッセージ用の遅延を考慮したツリーと長いメッセージ用のバンド幅を考慮したツリーを構築する．これらのスパニングツリーは，実行中にプロセッサが参加・脱退してもトポロジの変化に適応し，効率良く集合通信を行い続けることを可能にする．3 つから 4 つのクラスタに分散された 128 から 201 の実プロセッサにおいて，我々のブロードキャストの遅延は静的にトポロジを考慮した実装の 2 倍以内に収まり，バンド幅は静的にトポロジを考慮した実装の 82 パーセント出た．また，実行中に一部のプロセッサが参加または脱退した場合，我々のブロードキャストは，スパニングツリーが修復されるまで 8 秒程度性能が落ちたが，この間も正常に完了した．

Collective Operations for Wide-area Message Passing Systems Using Adaptive Spanning Trees

Hideo Saito,[†] Kenjiro Taura[†] and Takashi Chikayama[††]

We propose a method for wide-area message passing systems to perform collective operations using dynamically created spanning trees. In our proposal, broadcasts and reductions are performed efficiently using topology-aware spanning trees constructed at run-time; processors autonomously measure latency and bandwidth to create latency-aware trees for short messages and bandwidth-aware trees for long messages. Our spanning trees adapt to topology changes due to the joining or leaving of processors; when processors join or leave a computation, processors repair the spanning trees so that the effective execution of collective operations can continue. With 128 to 201 processors distributed over 3 to 4 clusters, the latency of our broadcast was within a factor 2 of a static topology-aware implementation, and our broadcast achieved 82 percent of the bandwidth of a static topology-aware implementation. Moreover, when some processors joined or left a computation, our broadcast temporarily performed poorly for about 8 seconds while the spanning trees adapted to the new topology, but completed successfully even during this time.

1. はじめに

近年，Wide-Area Network（WAN）においてメッセージパッシングが行われるようになってきた．WAN を用いると単一の Local-Area Network（LAN）を用いるのに比べてはるかに多くの計算資源を利用することができる．しかし，WAN で LAN 用のシステムをそのまま用いても，十分な性能を出せない．特に，LAN 用の集合通信は，プロセッサ間の遅延とバンド幅が一様という仮定に基づいて作られているため，まったく性能が出ない．WAN と LAN では遅延もバンド幅も何桁も異なるため，WAN 用の集合通信はネットワークトポロジを考慮して，高遅延・狭バンド幅のリンクを回避しなければならない．

これまでに，計算環境（用いられていれる計算資源やネットワーク性能）は変化しないと仮定して，トポロジを手動で与えることによって WAN で効率良く集合通信を行う手法がいくつか提案されている[3),4),6)~9)]．しかし，手動の設定はネットワーク性能などを詳細に知らなければ行えず，煩雑である．自動的に割り当てられた資源を用いる場合は，特に困難である．さらに，WAN における長時間にわたる大規模な計算では，動

† 東京大学大学院情報理工学系研究科電子情報学専攻
Department of Information and Communication Engineering, Graduate School of Information Science and Engineering, the University of Tokyo
†† 東京大学大学院新領域創成科学研究科基盤情報学専攻
Department of Frontier Informatics, Graduate School of Frontier Sciences, the University of Tokyo

的に計算環境の変化に対応できることが不可欠である．時間が経過するとネットワーク性能が変化するということだけでなく，他のユーザのために計算資源を解放したり，計算開始後に新たに利用可能になった計算資源を投入できたり，あるいは一部の計算資源が故障しても計算が続行できたりする必要があるからである．

ビデオストリーミングなどに用いられるコンテンツデリバリネットワーク（CDN: Content Delivery Network）用には，トポロジを考慮したオーバレイネットワークを動的に構築して，その上でアプリケーションレベルマルチキャストを行う手法がいくつも提案されている[1),2),5)]．しかし，コンテンツデリバリ用のマルチキャストはデータロスを許容したり，シングルソースを想定していたりして，メッセージパッシングの集合通信にそのまま用いることはできない．

そこで我々は，メッセージパッシング用に，動的に変化する環境で，設定なしで効率良く集合通信を行うための手法を提案する．ネットワーク性能の変動や故障によるプロセッサの脱退に対応することも視野には入れているが，本稿では意図的なプロセッサの参加・脱退のみを考える．

提案手法では，遅延を考慮したスパニングツリーとバンド幅を考慮したスパニングツリーを生成・維持し，それらに沿って集合通信を行う．この手法の特徴は以下のとおりである．

- 計算に用いられているプロセッサが変化していないときは効率が良い．
- プロセッサが参加・脱退している最中も正常に完了する．
- プロセッサの参加・脱退によるトポロジの変化に適応する．

我々は，2つのスパニングツリーアルゴリズムとそれらを用いるブロードキャストとリダクションをPhoenixメッセージパッシングライブラリ[12)]の拡張として実装した．また，複数のクラスタに分散された実プロセッサにおいて評価実験を行った．

以下，2 章で関連研究について述べる，3 章でPhoenix メッセージパッシングライブラリの説明をする．4 章で我々の提案する集合通信の実現方法を説明し，5 章で評価実験の結果を示す．最後に，6 章でまとめを行う．

2. 関 連 研 究

2.1 MPICH の集合通信

MPICH[11)] は最も広く用いられている Message Passing Interface（MPI）[10)] の実装である．

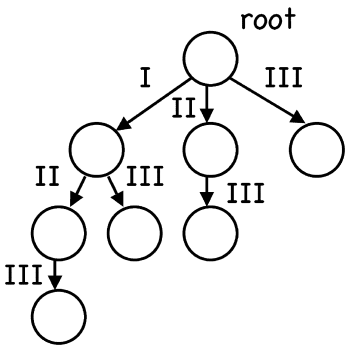


図 1 MPICH で短いメッセージのブロードキャストに用いられる binomial ツリー．8 プロセッサの場合，3 フェーズで行われる
Fig. 1 The binomial tree that MPICH uses for broadcasts of short messages. An 8-processor broadcast is performed in 3 phases.

MPICH は LAN 用に実装されているので，集合通信はプロセッサ間の遅延・バンド幅が等しいという仮定に基づいて設計されている[13)]．用いられるアルゴリズムは集合通信の種類やプロセッサの数によって異なるが，基本的なアイディアは木に沿ってメッセージを転送するというものである．短いメッセージには遅延を低くするような木が用いられ，長いメッセージにはバンド幅を広くするような木を用いられる．たとえば，短いメッセージのブロードキャストは binomial ツリーに沿って行われる（**図 1**）．一方，長いメッセージのブロードキャストでは，binomial ツリーを用いると同じプロセッサが同じデータを何度も送信しなければならずバンド幅を無駄に消費するので，いったんデータをすべてのプロセッサにスキャッタしてから，すべてのデータをすべてのプロセッサに集めるという手法が用いられる．

2.2 WAN における高性能な集合通信

Wide-area のリンクと local-area のリンクでは遅延もバンド幅も何桁も異なるため，MPICH のようにプロセッサ間の遅延・バンド幅が等しいと仮定したアルゴリズムを用いて WAN で性能を出すことはできない．そこで，ネットワークトポロジを考慮することによって WAN における集合通信の性能を上げるための研究がいくつか行われてきた[3),4),6)~9)]．

Kielmann らによって提案された MagPIe の集合通信では，クラスタ内とクラスタ間で異なる木に沿って通信を行うことによって，データが不必要に wide-area リンクを通ることを回避する[9)]．クラスタ間の通信では各クラスタを coordinator ノードと呼ばれるノードが代表する．たとえば，ブロードキャストでは，ルートが各クラスタの coordinator ノードにメッセージを送

信した後，各 coordinator ノードが自分のクラスタ内で MPICH のような binomial ツリーを用いたブロードキャストを行う．

MagPIe は遅延にしか注目していないので短いメッセージにしか有効でないが，Kielmann らはバンド幅を考慮した集合通信も提案している[8]．その手法では，事前に測定した遅延・バンド幅をシステムに入力として与え，メッセージサイズに応じて最適な木の形状とメッセージのセグメントサイズを算出する．

MPICH-G2 は MPICH をグリッド用に拡張したものである[7]．MPICH-G2 の集合通信は，トポロジを wide-area，local-area，system-area☆などと複数の階層に分けるため，MagPIe のようにクラスタ内とクラスタ間の 2 階層だけの区別をするよりさらに性能が出る[6]．

これらの研究は，計算環境が静的であると仮定しているという点と，事前にネットワークに関する情報を与える必要があるという点で，我々の提案する手法と異なる．

2.3 CDN 用アプリケーションレベルマルチキャスト

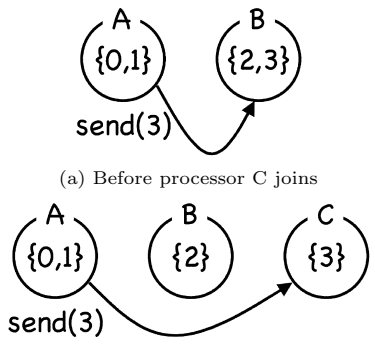
ビデオストリーミングなどのコンテンツデリバリ用に，トポロジを考慮したオーバレイネットワークを動的に構築して，その上でアプリケーションレベルマルチキャストを行う手法がいくつも提案されている[1),2),5)]．しかし，これらはコンテンツデリバリ用に設計されているため，メッセージパッシングアプリケーションの要求を満たしていない．

Banerjee らは低オーバヘッドで遅延を考慮したオーバレイネットワークを動的に構築・維持する手法を提案しており，これを用いることによって効率良く短いメッセージをマルチキャストできる[1)]．遅延を考慮して構築したオーバレイネットワークを長いメッセージに用いる方法も提案しているが，用いるツリーのファンアウトが大きすぎるのと，実際にバンド幅を測定するわけではないので，長いメッセージ用に特化した手法ほどの性能は期待できない．また，データストリームアプリケーションを想定しているのでデータロスを許容するが，メッセージパッシングではデータロスは許されない．

Overcast はバンド幅を考慮したオーバレイネットワークを動的に構築・維持することによって大容量のデータを効率良くマルチキャストする[5)]．ビデオストリーミング用に設計されているのでシングルソースであるが，メッセージパッシングではすべてのノードが集合通信のルートになれる必要がある．

さらに，コンテンツデリバリで用いられるマルチキャストは 1-to-N の操作であるが，メッセージパッシングでは 1-to-N の操作以外に N-to-1 の操作や N-to-N の操作も行える必要がある．

☆ 同一ホスト内



(a) Before processor C joins

(b) After processor C joins

図 2 Phoenix の仮想ノード．(a) では仮想ノード 3 はプロセッサ B にマップされているので，仮想ノード 3 宛に送信したメッセージはプロセッサ B に届く．一方，(b) のようにプロセッサ C が新たに参加して仮想ノード 3 をプロセッサ B から受け継ぐと，仮想ノード 3 宛のメッセージはプロセッサ C に届くようになる

Fig. 2 Virutal nodes in Phoenix. In (a), virtual node 3 is mapped to processor B, so a message addressed to virtual node 3 arrives at processor B. In (b), processor C has joined the computation and acquired virtual node 3 from processor B—now, a message addressed to virtual node 3 arrives at processor C.

3. Phoenix

本章では，我々の実装の基となる Phoenix メッセージパッシングライブラリ[12)] について説明する．

Phoenix はグリッド用のメッセージパッシングライブラリであるが，MPI の実装ではなく，独自の API を持っている．この API は一般的なメッセージパッシング用に設計されているが，計算資源の動的な増減を支援しているところに特徴がある．

Phoenix ではメッセージは MPI のようにプロセッサ宛に送信されるのではなく，仮想ノード宛に送信される．プロセッサと仮想ノードの間には任意の $1:n$ のマッピングがあり，このマッピングはプログラムの実行中に変更できる．Phoenix は point-to-point 通信用のルーティングテーブルを維持し，メッセージは配達時に仮想ノードがマップされているプロセッサに届けられる（**図 2** (a)）．

アプリケーションは，実行時にプロセッサと仮想ノードのマッピングを変更することによって，プロセッサの動的な参加・脱退を支援することができる（図 2 (b)）．

計算に参加するには，プロセッサはすでに参加している他のプロセッサに接続し，そのプロセッサにマップされている仮想ノード集合の一部を譲ってもらう．計算から脱退するには，プロセッサは自分にマップされている仮想ノードを他のプロセッサに譲ってから，脱退する．

Phoenix では計算資源が動的に変化するため，集合通信を既存の手法を用いて行うことはできない．MagPIe や MPICH-G2 のように計算開始時の情報を用いて生成した木で集合通信を行うと，計算開始後に参加したプロセッサにメッセージが届かない．また，計算開始後にプロセッサが脱退すると，そのプロセッサの子孫にメッセージが届かなくなってしまう．

4. 提案手法

4.1 概　　要

我々の提案する集合通信では，トポロジを意識したスパニングツリーを動的に生成・維持し，それらのツリーに沿ってブロードキャストやリダクションを行う．遅延によって性能が決まる短いメッセージとバンド幅によって性能が決まる長いメッセージを統一的に扱うのは困難なので，短いメッセージ用には遅延を考慮したスパニングツリーを生成し，長いメッセージ用にはバンド幅を考慮したスパニングツリーを生成する．また，各プロセッサがブロードキャストやリダクションのルートになれる必要があるので，各プロセッサをルートとするスパニングツリーをプロセッサの数だけ生成する．

各スパニングツリーは，子が適切な親を見つけるという形態で生成する．そのために，各プロセッサは，ランダムに選択した他のプロセッサとの遅延・バンド幅を測定し，各ツリーにおいてそのプロセッサが現在の親より適切かどうかを検討する．遅延の測定には 1 バイトのメッセージを，バンド幅の測定には 128 キロバイトのメッセージを実際に転送する．

測定・親探しは，プロセッサが参加・脱退したとき（アプリケーション起動時も含む）のみ行い，計算に用いられているプロセッサが変化していないときは行わない．プロセッサが参加・脱退したとき，各プロセッサはある一定の数のランダムに選択したプロセッサと測定を行い，ツリーを更新する．この数は多ければ多いほどトポロジをよく反映したツリーができるが，選択したプロセッサの中に同一 LAN のものがいくつかあればそれなりに良いツリーができる．本稿の実装では，各プロセッサは 10 個のプロセッサをランダムに選択し，測定を行う．

以下，4.2 節と 4.3 節で 2 つのスパニングツリーアルゴリズムを説明し，4.4 節と 4.5 節で生成したスパニングツリーを用いてブロードキャストとリダクションを行う方法を説明する．

4.2 遅延を考慮したスパニングツリー

遅延を考慮したスパニングツリーアルゴリズムでは，以下のような性質を満たすツリーを生成・維持することを目指す．

- LAN をまたぐ親子関係が少ない．
- LAN 内の親子関係はランダム．

各プロセッサは，まだ親のいないツリーにおいてルートにつながっているプロセッサを見つけると，ただちにそのプロセッサをそのツリーにおける親にする．また，すでに親のいるツリーにおいてもより良い親が見つかると親を乗り換える．より良い親であるかどうかの判断には，自分と現在の親の往復時間（RTT: Round-Trip Time），自分と親候補の往復時間，自分とルートのツリーに沿った距離，親候補とルートのツリーに沿った距離の 4 つの値を用いる．具体的には，プロセッサ p は，親子候補 $cand$ が以下の 2 つの条件を両方満たした場合，$cand$ の子になる．

(1) $RTT_{p,cand} < RTT_{p,parent}$
自分と親候補の RTT が自分と現在の親の RTT より短い．

(2) $dist_{cand,root} < dist_{p,root}$
親候補とルートの距離が自分とルートの距離より短い．

ここで，親・親候補との RTT は直接 ping-pong を行うことによって測定し，ルートとの距離はルートからツリーに沿って 1 ホップ分ずつ RTT を加えたものを用いる（**図 3**）．親候補とルートの距離は親候補に伝えてもらう必要があるが，これはバンド幅測定に用いる 128 キロバイトの中に含める．

プロセッサが参加・脱退した直後のスパニングツリーは，2 つの LAN をまたぐ親子関係を多数持ち，LAN の中でも極端に深かったり浅かったりする可能性がある．しかし，条件 (1) により各プロセッサはより近いプロセッサに親を乗り換えていくので，LAN をまたぐ親子関係は次々と LAN 内の親子関係に置き換えられていく．

また，極端に深いツリーができてしまっても，それはすぐにより浅いツリーになる．これを説明するために，LAN 内のすべてのプロセッサ間の遅延が等しいとする．すると，あるプロセッサが LAN 内に親候補を選択した場合，条件 (1) は 1/2 の確率で満たされる．また，このプロセッサは深い位置にいるので，LAN 内

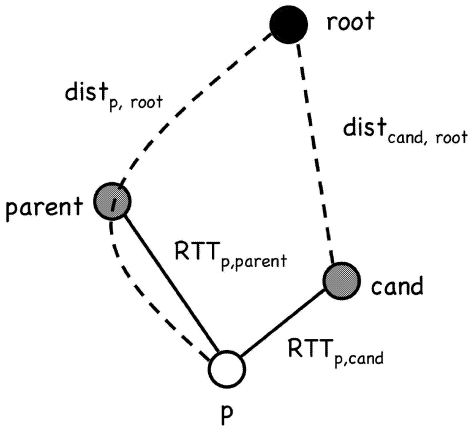


図 3 親・親候補との RTT は直接 ping-pong を行うことによって求める．ルートとの距離はルートからツリーに沿って 1 ホップ分ずつ RTT を加えたものを用いる

Fig. 3 The RTT between a processor and its parent or a parent candidate is determined by directly performing a ping-pong. A processor's distance from the root is computed by adding the RTT of each hop from the root to that node along the spanning tree.

のほとんどのプロセッサは自分より浅い位置におり，条件 (2) も高い確率で満たされる．したがって，ツリーの極端に深い位置に付いてしまったプロセッサは，数回同一 LAN 内に親候補を選択すればより浅い位置に移動することができる．

極端に浅いツリーができてしまった場合も，それはすぐにより深いツリーになる．深いツリーの場合と同様に，あるプロセッサが LAN 内に親候補を選択した場合，条件 (1) は 1/2 の確率で満たされる．また，浅いツリーでは浅い位置にいるプロセッサには多くの兄弟がおり，兄弟を親候補に選択すれば条件 (2) も 1/2 の確率で満たされる．したがって，浅いツリーで浅い位置にいるプロセッサは，数回同一 LAN 内に親候補を選択すればより深い位置に移動することができる（**図 4**）．

各プロセッサが各スパニングツリーに関して保持する情報は以下のとおりである．

- 親プロセッサ（1 つ）
- 子プロセッサ（複数）
- $RTT_{p,parent}$
- $dist_{p,root}$
- それぞれの子プロセッサにおいて，そのプロセッサとその子孫にマップされている仮想ノードの集合

子プロセッサに関する情報はスパニングツリーの生成には必要ないが，実際にブロードキャストやリダクションを行う際に必要である．

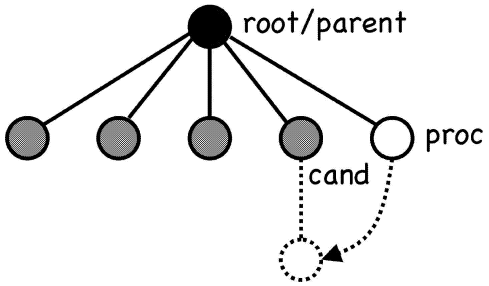


図 4 浅いツリーでは浅い位置にいるプロセッサは兄弟の子になることによってより深い位置に移動することができる

Fig. 4 In a shallow tree, a processor in a shallow position is able to move to a deeper position by becoming a child of one of its children.

4.3 バンド幅を考慮したスパニングツリー

バンド幅を考慮したスパニングツリーアルゴリズムの目的は，長いメッセージのブロードキャストでバンド幅を効率良く利用できるツリーを生成・維持することである．そのために，以下の性質を満たすツリーを構築する．

(1) ルートから各ノードへのツリーに沿ったバンド幅が広い．
(2) 枝分かれが少ない．

性質 (1) だけでも point-to-point 通信のバンド幅は広くなるが，ブロードキャストのバンド幅を広くするためには性質 (2) も必要である．なぜならば，親が複数の子にメッセージを転送すると，それらの子がネットワーク資源（親のネットワークインタフェース，親と子の間のリンク・スイッチ）を共有しなければならないからである．

親候補の適性を判断するには，実際にルートからツリーに沿ってメッセージをダウンロードするのが最も正確であるが，それではルートにかかる負荷が大きい．そこで，我々のアルゴリズムでは，各プロセッサは，自分と親候補の間のやりとりから得られる情報のみで親候補の適性を判断する．つまり，プロセッサ p は，以下の情報を用いて，ブロードキャスト時に親候補 *cand* からデータを受信する速さ $est_{p,cand}$ を見積もる．

- bw_{p2p}：p と *cand* の間の point-to-point のバンド幅
- $n_{children}$：*cand* の子の数（p が *cand* の子になった場合の p の兄弟の数）
- est_{cand}：ブロードキャスト時に *cand* がその親からデータを受信する速さ

P は，*cand* から 128 キロバイトをダウンロードするのにかかる時間を測定することによって bw_{p2p} を

求める．また，est_{cand} と $n_{children}$ は，$cand$ からダウンロードする 128 キロバイトの中に含めてもらう．これらの情報を用いて，p は e_{cand} を次式によって求める．

$$est_{p,cand} = min(est_{cand}, bw_{p2p}/n_{children} + 1)$$

$Est_{p,cand}$ が p が現在の親からデータを受信する速さ $est_{p,parent}$ より大きい場合，p は $cand$ に親を乗り換える．

LAN では親のインタフェースがボトルネックになることが多いため，point-to-point のバンド幅を子の数で割るという我々の手法はそれなりに正確である．一方，WAN では wide-area リンクがボトルネックになることが多いため，我々の手法はあまり正確でない．しかし，それでも我々の手法は枝分かれの数を少なくするという効果は持つ．

各プロセッサが各スパニングツリーに関して保持する情報は以下のとおりである．

- 親プロセッサ（1 つ）
- 子プロセッサ（複数）
- $est_{p,parent}$
- それぞれの子プロセッサにおいて，そのプロセッサとその子孫にマップされている仮想ノードの集合

4.4 ブロードキャスト

4.2 節・ 4.3 節のアルゴリズムを用いて生成したスパニングツリーを用いてブロードキャストを行う方法を説明する．MPI におけるブロードキャストは全プロセッサにメッセージを届ける操作であるが，Phoenix におけるブロードキャストは全仮想ノードにメッセージを届ける操作になる．ただし，同一のプロセッサにマップされている複数の仮想ノードにはまとめてメッセージを届けてもよいことにする．

短いメッセージのブロードキャストには 4.2 節のアルゴリズムによって生成した遅延を考慮したスパニングツリーを用い，長いメッセージのブロードキャストには 4.3 節のアルゴリズムによって生成したバンド幅を考慮したスパニングツリーを用いる．同じ長さのメッセージでもトポロジによってどちらのツリーの方が性能が出るかは変わるが，本稿では 256 KB 未満のメッセージでは遅延を考慮したツリーを，256 KB 以上のメッセージではバンド幅を考慮したツリーを用いる．

スパニングツリーが安定しているときは，スパニングツリーに沿ってルートから葉に向けてメッセージを転送するだけでよい．しかし，スパニングツリーが変化しているときは，スパニングツリーに沿ってメッセージを転送しても全仮想ノードにメッセージが届くとは限らない．たとえば，ブロードキャスト中にプロセッサが脱退するとそのプロセッサの子孫にマップされている仮想ノードにメッセージが届かなくなってしまう．また，ブロードキャスト中に仮想ノードのマッピングが変わった場合も，全仮想ノードにメッセージが届く保証はない．

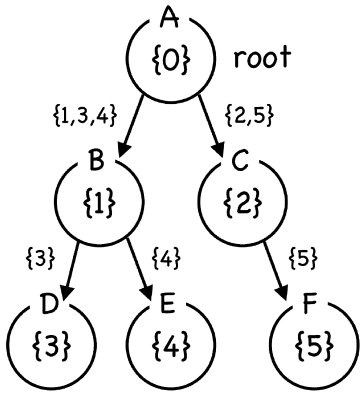
そこで我々の提案するブロードキャストでは，メッセージのヘッダにまだメッセージが届けられていない仮想ノードを列挙しておく．ルートはブロードキャストを開始する際，それぞれの子プロセッサを経由してメッセージが届けられるべき仮想ノードを列挙しておく．親からメッセージを受け取ったプロセッサは，ヘッダに含まれている仮想ノードを見て，以下の処理を行う．

(1) 自分にマップされているものがある場合は，メッセージをユーザプログラムに届ける．
(2) 子を経由して届けられるものがある場合は，ヘッダにそれらの仮想ノードを列挙したメッセージを子に転送する．
(3) (1) にも (2) にも該当しない仮想ノードに関しては point-to-point 通信を用いてメッセージを送信する．

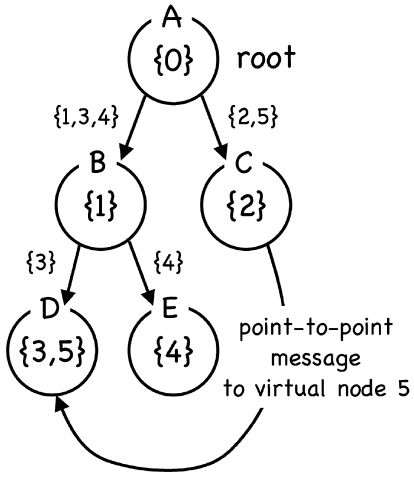
スパニングツリーが安定しているときは，ヘッダに列挙されている仮想ノードがスパニングツリーと一致するので，(1) と (2) のみによる効率の良いブロードキャストが行われる（**図 5** (a)）．一方，スパニングツリーが変化しているときは，ヘッダに列挙されている仮想ノードがスパニングツリーと一致しないため，(3) が行われることがある（図 5 (b)）．Point-to-point 通信を用いるとブロードキャストの効率は落ちるが，全仮想ノードにメッセージが届くことを保証できる．スパニングツリーの変化が終わると，ヘッダに列挙されている仮想ノードがスパニングツリーと一致するようになり，(3) は行われなくなる．

4.5 リダクション

Phoenix におけるリダクションは，各仮想ノードにマップされているデータに対して和を求める・最大値を求めるなどの演算を行い，その結果を 1 つの仮想ノードに届けるというものである．ツリーに沿って行うリダクションは，葉プロセッサが自分にマップされている仮想ノードにマップされているデータをツリーの上方へ送ることによって開始する．仮想ノードが複数マップされているプロセッサは複数のデータを送る．データはツリーに沿って葉からルートへと集められていき，最終的にすべてのデータがルート仮想ノードがマップされているプロセッサに集まった時点で，その



(a) Stable Tree



(b) Changing Tree

図 5 (a) ではスパニングツリーが安定しているので，メッセージはスパニングツリーに沿って効率良く転送される．(b) では，プロセッサ A がプロセッサ C にメッセージを転送した直後にプロセッサ F がプロセッサ D に仮想ノード 5 を譲って脱退した．プロセッサ C は仮想ノード 5 を転送する子がいないので，point-to-point 通信で仮想ノード 5 宛のメッセージを送る

Fig. 5 In (a), the spanning tree is stable, so messages are forwarded efficiently along the spanning tree. In (b), processor F has given virtual node 5 to processor D just after processor A forwarded a message to processor C. As processor C does not have a child to forward virtual node 5 to, it sends a point-to-point message to virtual node 5.

プロセッサのユーザプログラムに届けられる．

スパニングツリーが安定しているときは，各プロセッサはすべての子プロセッサからのデータを待った後，親プロセッサにメッセージを転送すればよい．一方，スパニングツリーが変化しているときは，すでに他のプロセッサにメッセージを転送したプロセッサを待ってしまう可能性がある．

そこで我々の提案するリダクションでは，タイムアウトを用い，子プロセッサをある一定の時間しか待たない．各プロセッサはリダクション関数が呼ばれたときにタイムアウトを設定し，タイムアウトが起こったらデータを送ってきていないプロセッサがいても，すでに受け取ったデータを親に転送する．親プロセッサへデータを転送した後に送られてきたデータや子でないプロセッサから送られてきたデータはただちに転送する．

タイムアウトは，スパニングツリーが安定しているときにはタイムアウトが起こる前にすべての子プロセッサからデータが届く程度の長さに設定しなければならない．これは，タイムアウトをリダクションルートとそれから最も遠い葉の間の時間より長くすることによって実現できるが，各プロセッサはこの値を知らず，この値を集合的に算出するためには多くの通信を要する．しかし，各プロセッサはあらゆるスパニングツリーにおける自分とルートの時間を知っているので，この値の最大値を近似値として用いる．

5. 評価実験

本章では，提案手法を評価するために行った実験の結果を示す．実験は，それぞれ 3 つか 4 つのクラスタに分散された 128 から 201 の実プロセッサにおいて行った．Phoenix の point-to-point 通信を用いて測定した RTT は，クラスタ内で数百マイクロ秒，クラスタ間では数ミリ秒であった．以下，5.1 節と 5.2 節でスパニングツリー安定時の性能について述べ，5.3 節でプロセッサ参加・脱退時の挙動について述べる．

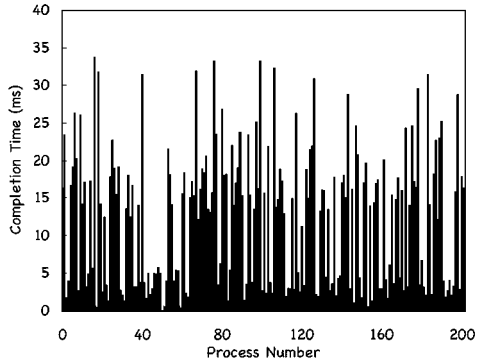
5.1 スパニングツリー安定時のブロードキャスト性能

最初の実験では，スパニングツリー安定時のブロードキャスト性能を評価するために，各プロセッサが遅延とバンド幅の測定を終えて，スパニングツリーが安定してから，ブロードキャストを行った．仮想ノードは，各プロセッサに 1 つだけマップした．

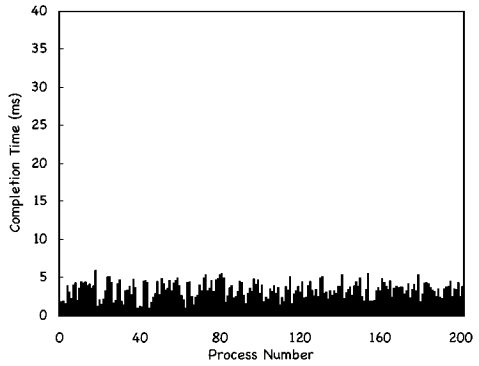
5.1.1 短いメッセージ

短いメッセージの評価のためには，1 バイトのブロードキャストが各プロセッサに届くのに要した時間を測定した．比較対象として，MPICH のアルゴリズムを用いたトポロジを考慮しなかった実装を用いた場合と，MagPIe のアルゴリズムを用いた静的にトポロジを考慮した実装の場合についても，測定を行った．3 つのクラスタに分散された 201 プロセッサを用いた場合の結果を**図 6** に示す．

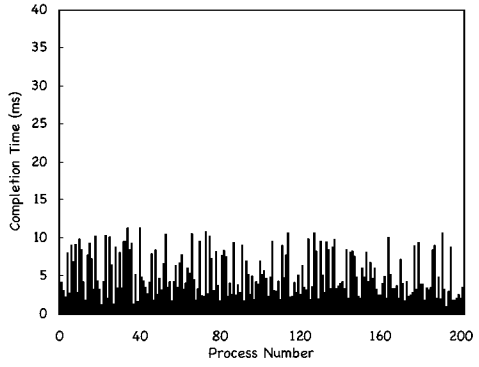
トポロジを考慮しなかった実装では，LAN 内の通信のみでメッセージが届いたプロセッサはルートがブロードキャストを開始してから数ミリ秒以内にメッ



(a) MPICH-like Implementation



(b) MagPIe-like Implementation



(c) Our Implementation

図 6 3 つのクラスタに分散された 201 プロセッサにおける 1 バイトのブロードキャスト
Fig. 6 1-byte broadcast over 201 processors in 3 clusters.

セージを受け取っているが，LAN 間の通信を何度も経てからメッセージが届いたプロセッサは 30 ミリ秒以上待たされている（図 6 (a)）．

一方，静的にトポロジを考慮した実装では，不必要に wide-area リンクを通るメッセージがないため，最も待たされたプロセッサでも 6 ミリ秒後にはメッセージが届いており，トポロジを考慮しなかった場合より 7 倍近く速い（図 6 (b)）．これは Kielmann らの結果と一致する．

我々の提案手法を用いたブロードキャストでは，最も待たされたプロセッサは 11 ミリ秒後にメッセージを受信しており，トポロジを考慮しなかった場合に比べて 3 倍速い（図 6 (c)）．一方，静的にトポロジを考慮した場合と比べると 2 倍近く遅くなってしまっている．主な理由としては，生成されたツリーの LAN 内におけるサブツリーが binomial ツリーでないということと，ヘッダに仮想ノードを列挙するのにオーバヘッドがともなうということがある．Binomial ツリーではすべての葉に同時にメッセージが届くが，我々のツリーでは葉にメッセージが届くタイミングはばらばらである．

5.1.2 長いメッセージ

長いメッセージの評価のためには，32 キロバイトから 64 メガバイトのブロードキャストにおけるバンド幅を測定した．バンド幅の値には，送信バイト数とプロセッサ数の積（必要最小の通信量）を時間で割った値を用いた．また，比較対象として，提案手法を用いた場合（Dynamic）以外に，トポロジを考慮しなかった場合（MPICH-like）と，静的にトポロジを考慮した場合（MagPIe-like と List）についても，測定を行った．Dynamic では，256 キロバイト未満のメッセージでは遅延を考慮したツリーを用い，256 キロバイト以上のメッセージではバンド幅を考慮したツリーを用いた．MPICH-like では，MPICH の長いメッセージ用のブロードキャストのアルゴリズムを用いた．MagPIe-like では，MagPIe のようにルートが各クラスタの coordinator ノードにメッセージを送った後，各クラスタの中で MPICH の長いメッセージ用のブロードキャストと同じことを行った．List では，ボトルネックが最も広くなるようにプロセッサを 1 本のリストに並べ，そのリストに沿ってブロードキャストを行った．MagPIe-like のアルゴリズムも List のアルゴリズムもバンド幅を意識しているが，List では同じプロセッサが同じデータを送信することがいっさいないので，非常に長いメッセージでは List の方が MagPIe-like より速い．

図 7 に，4 つのクラスタに分散された 137 プロセッサを用いた場合の結果を示す．トポロジを考慮しなかった MPICH-like は，トポロジを考慮した他の 3 つの実装に比べて著しく悪い性能を示した．我々の提案手法を用いた実装は，512 キロバイト以下のメッセージでは MagPIe-like に，8 メガバイト以上のメッセージでは List に劣るが，全体的にそれなりに良い性能が出ており，バンド幅を考慮した結果性能が上がっていることは明らかである．我々の実装で List ほどは広

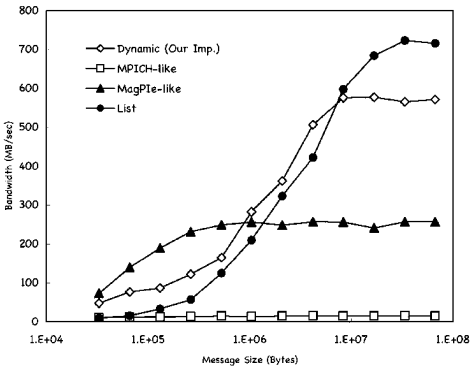


図 7 4 つのクラスタに分散された 137 プロセッサにおけるブロードキャスト
Fig. 7 Broadcast over 137 processors distributed over 4 clusters.

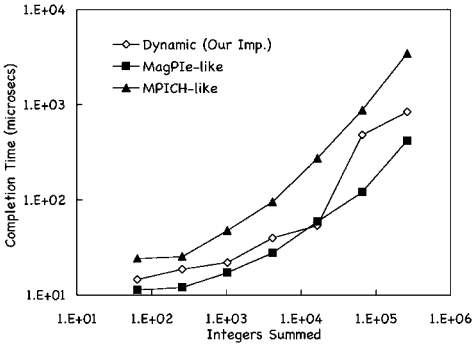


図 8 3 つのクラスタに分散された 128 プロセッサにおけるリダクション
Fig. 8 Reduction over 128 processors distributed over 3 clusters.

いバンド幅が出なかった主な理由は，我々の実装ではバンド幅を意識はしているものの，List のように枝分かれがいっさいないわけではないということである．

5.2 スパニングツリー安定時のリダクション性能

次の実験では，スパニングツリー安定時のリダクション性能を評価した．この実験では，各プロセッサに仮想ノードを 1 つずつマップし，同じ長さの配列を与え，配列の各要素の和を求めるのにかかる時間を測定した．**図 8** に 3 つのクラスタに分散された 128 プロセッサを用いた場合の結果を示す．ブロードキャストの場合と同様に，我々の提案手法を用いた実装は，トポロジを考慮しなかった実装よりは速かったが，静的にトポロジを考慮した実装よりは遅かった．

5.3 プロセッサ参加・脱退時の挙動

最後の実験では，我々の提案手法を用いた集合通信のプロセッサ参加・脱退時の挙動について調べるために，繰り返し 4 メガバイトのブロードキャストを行っている最中にプロセッサの一部を脱退・再参加させた．

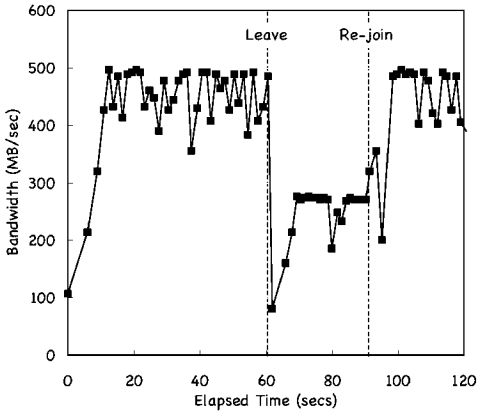


図 9 繰り返し 4 メガバイトのブロードキャストを行っている最中にプロセッサが脱退・再参加したときのブロードキャストのバンド幅
Fig. 9 Broadcast bandwidth when some processors left and then rejoined while 4-megabyte broadcasts were continually being performed.

最初は各プロセッサに仮想ノードを 1 つずつマップしておき，途中で以下のようにマッピングを変更した．

(1) アプリケーション起動 60 秒後に，半分のプロセッサが残りのプロセッサに仮想ノードを譲渡．
(2) その 30 秒後に，(1) で仮想ノードを譲渡されたプロセッサが仮想ノードを返却．

つまり，アプリケーション起動 60 秒後から 30 秒間，半分のプロセッサに仮想ノードが 2 つずつマップされている．

図 9 に 4 つのクラスタに分散された 160 プロセッサを用いた場合の結果を示す．アプリケーション起動直後は，スパニングツリーが生成中であったため，ブロードキャストのバンド幅は狭い．しかし，アプリケーション起動 11 秒後にはスパニングツリーが安定したため，ブロードキャストのバンド幅も安定した．

アプリケーション起動 60 秒後には，半分のプロセッサが脱退したため，スパニングツリーが壊れ，ブロードキャストの性能は大きく低下した．しかし，ツリーは 8 秒後には修復され，ブロードキャストの性能も再び安定した．脱退したプロセッサが再参加したときにも，似たような挙動が見られた．

このように，我々の提案手法では，計算中にプロセッサが参加・脱退すると，集合通信の性能は一時的に低下するが，メッセージはすべてのプロセッサに届き，スパニングツリーが再び安定すると新しいトポロジを考慮した集合通信が行われるようになる．

6. おわりに

本稿では，広域メッセージパッシングシステム用に，

動的に生成したスパニングツリーを用いて集合通信を行う手法について述べた．実行時に遅延・バンド幅を測定してスパニングツリーを生成・維持するアルゴリズムを説明し，生成したツリーを用いてブロードキャストとリダクションを行う手法を説明した．

評価実験では 3 つから 4 つのクラスタに分散された 128 から 201 の実プロセッサを用いて，以下のことを示した．

(1) スパニングツリーが安定しているときは，提案手法を用いたブロードキャスト・リダクションが，静的にトポロジを考慮した実装ほどではないがトポロジを考慮しなかった実装よりは確実に速かった．1 バイトのブロードキャストでは遅延が静的にトポロジを考慮した実装の 2 倍以内に収まり，64 メガバイトのブロードキャストでは静的にトポロジを考慮した実装のバンド幅の 82 パーセントを達成することができた．

(2) 計算中にプロセッサが参加・脱退した場合，提案手法を用いた集合通信はスパニングツリーが新しいトポロジに適応する間 8 秒程度性能が低下したが，その間も正常に完了した．

今後の課題には，提案したブロードキャスト・リダクションの最適化によって静的にトポロジを考慮した実装との性能差を縮めるということがある．たとえば，提案したブロードキャストではメッセージを転送する子の順番を考慮していないが，子孫にマップされている仮想ノードが多い子から順に転送することによって性能が上がることが期待される．また，遅延を考慮したスパニングツリーアルゴリズムでは，RTT だけではなく，転送オーバヘッドも考慮することによって，より正確にネットワークをモデル化できる．さらに，ヘッダに列挙する仮想ノードの形式を工夫することによってオーバヘッドを削減することもできる．

また，本稿で提案したアルゴリズムでは各プロセッサは他のプロセッサとほとんど情報を共有しないが，ある程度情報を共有することによってより効率良くスパニングツリーを構築できるはずである．たとえば，現在のアルゴリズムでは親候補はランダムに選択しているが，他のプロセッサからヒントをもらえれば，近いプロセッサを優先的に選択できるかもしれない．

参 考 文 献

1) Banerjee, S., Bhattacharjee, B. and Kommareddy, C.: Scalable Application Layer Multicast, *Proc. 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp.205–217 (2002).
2) Bozdog, A., van Renesse, R. and Dumitriu, D.: SelectCast — A Scalable and Self-Repairing Multicast Overlay Routing Facility, *Proc. 2003 ACM Workshop on Survivable and Self-Regenerative Systems*, pp.33–42 (2003).
3) Gabriel, E., Resch, M., Beisel, T. and Keller, R.: Distributed Computing in a Heterogeneous Computing Environment, *Proc. 5th European PVM/MPI User's Group Meeting*, pp.180–187 (1998).
4) Husbands, P. and Hoe, J.C.: MPI-StarT: Delivering Network Performance to Numerical Applications, *Proc. 1998 ACM/IEEE Conference on Supercomputing*, pp.1–15 (1998).
5) Jannotti, J., Gifford, D.K., Johnson, K.L., Kaashoek, M.F. and O'Toole, J.J.W.: Overcast: Reliable Multicasting with an Overlay Network, *Proc. 4th Symposium on Operating System Design and Implementation*, pp.197–212 (2000).
6) Karonis, N., de Supinski, B., Foster, I., Gropp, W., Lusk, E. and Bresnahan, J.: Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance, *Proc. 14th International Symposium on Parallel and Distributed Processing*, pp.377–384 (2000).
7) Karonis, N.T., Toonen, B. and Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing*, Vol.63, No.5, pp.551–563 (2003).
8) Kielmann, T., Bal, H.E. and Gorlatch, S.: Bandwidth-Efficient Collective Communication for Clustered Wide Area Systems, *Proc. 14th International Symposium on Parallel and Distributed Processing*, pp.492–499 (2000).
9) Kielmann, T., Hofman, R.F.H., Bal, H.E., Plaat, A. and Bhoedjang, R.A.F.: MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems, *Proc. 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp.131–140 (1999).
10) Message Passing Interface (MPI) Forum. http://www.mpi-forum.org
11) MPICH-A Portable Implementation of MPI. http://www-unix.mcs.anl.gov/mpi/mpich
12) Taura, K., Endo, T., Kaneda, K. and Yonezawa, A.: Phoenix: A Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources, *Proc. 9th ACM SIGPLAN Symposium on Principles and*

Practice of Parallel Programming, pp.216–229 (2003).
13) Thakur, R. and Gropp, W.: Improving the Performance of Collective Operations in MPICH, *Proc. Euro PVM/MPI*, pp.257–267 (2003).

(平成 17 年 1 月 24 日受付)
(平成 17 年 6 月 21 日採録)

斎藤　秀雄（正会員）
1981 年生．2004 年東京大学工学部電子情報工学科卒業．同年 4 月より東京大学大学院情報理工学系研究科電子情報学専攻修士課程在学中．

田浦健次朗（正会員）
1969 年生．1997 年東京大学大学院理学博士（情報科学専攻）取得．1996 年より東京大学大学院理学系研究科情報科学専攻助手．2001 年東京大学大学院情報理工学系研究科電子情報学専攻講師．2002 年より同助教授．

近山　　隆（正会員）
1953 年生．1977 年東京大学工学部計数工学科卒業．1982 年東京大学大学院情報工学専門課程修了．工学博士取得．同年より ICOT において第五世代コンピュータプロジェクトに参加．1995 年より東京大学に移り，現在同新領域創成科学研究科基盤情報学専攻教授．