高速なトポロジ推定 ——ネットワークを考慮した並列計算のための基盤として

白井達也[†] 斎藤秀雄[†] 田浦健次朗[†]

頻繁に通信を行う並列アプリケーションの性能向上のためには，ネットワークを考慮した最適化が非常に重要である．そのためには LAN 内であっても複数スイッチの構成情報を得る必要がある．しかしホストが頻繁に増減する動的な環境，より仮想化された環境ではネットワークの情報を手動で設定することは大きな手間になる．また，既存の研究は目的に特化した情報を用いるものが主流である．本稿ではより汎用的な情報としてネットワークトポロジのツリー表現を扱い，エンドホスト間の RTT だけを用いてツリーを推定する手法を提案する．我々の推定手法は特別なプロトコルを用いず，ネットワークへの負荷が低く，高速であることを特徴とする．本稿の手法は 1 クラスタ 64 ホストのトポロジを 4 秒程度で，4 クラスタ 256 ホストのトポロジを 15 秒程度で推定した．また高いスケーラビリティを得られることが分かった．さらに推定したトポロジを用いてバンド幅マップの構築のための効果的な並列化，および長いメッセージのブロードキャストの最適化を実現した．

A Fast Topology Inference ——A Building Block for Network-aware Parallel Processing

Tatsuya Shirai,[†] Hideo Saito[†] and Kenjiro Taura[†]

Adapting to network is the key to achieving high performance for communication-intensive applications. For open and dynamic environments whose resource pools change frequently, it will be tedious to supply network information. Also, previous systems often distinguish inter-cluster from intra-cluster links. Finally, the system obtains information for a certain purpose. In this paper we propose a framework in which the system infers the topology of the network as a tree. The heart of our proposal is a portable and non-intrusive algorithm that quickly and automatically infers such a topology. Our system built a topology of 64 hosts in a single cluster in about 4 seconds and 256 hosts in 4 clusters in 15 seconds in our experimental environment. And we implemented 2 applications using the inferred topology: (1) construction of bandwidth maps and (2) optimized broadcast of large messages.

1. はじめに

多くの集合通信や通信を頻繁に行うアプリケーションの性能は，物理ネットワーク上の通信パターンによって変動する．特に多数のスイッチや複数の LAN を用いた大規模なネットワークでは，この影響は非常に重要になる．たとえばグリッド環境でのブロードキャストは，クラスタ間の通信を最小にすることが重要である[12]．集合通信やアプリケーションの最適化に関する研究は様々なものがあるが，その多くが個々の処理について細かい設定を手動で与えるものである．たとえばグリッド環境で動作する MPI に関する多くの研究[1),9)~11)] では，MPI のシステムが用いるホストのグループ分けが与えられているということを仮定する．しかし，管理者が異なる複数のホスト群（たとえばマルチクラスタ環境）で各ユーザが異なるホスト集合を用いる場合，あるいは使えるホスト群が頻繁に変更されるという，よりオープンで動的な環境，さらにはより仮想化された環境ではホストのグループ分けを手動で与えることは大きな手間になる．また，これらのシステムはパターン化された通信について最適化されたものしか提供せず，プログラマが実装する通信パターンの最適化には用いられない．

本稿では，このような最適化の基盤となるネットワーク情報の表現方法と，その推定手法について述べる．我々はネットワークをツリーモデルで表現し，手動設定を用いずにそのツリーを推定する手法を提案する．我々の推定手法は非常に高速で，ネットワークへの負荷が小さく，ネットワーク的にヘテロな環境で動

† 東京大学
The University of Tokyo

作する．推定アルゴリズムに必要な情報は推定対象のホストの IP アドレスとポート番号だけで，それらを接続するスイッチの情報は必要としない．推定されたツリーは遅延の情報をリンクの重みとして有する．また本稿では推定したトポロジを用いた最適化の例として，バンド幅マップの効率的な構築方法と，長いメッセージのブロードキャストの最適化方法を示す．

現実のネットワークは循環構造を持つことがあり，それをツリーで表現することは議論の余地がある．しかし，ネットワーク情報の抽象化と一般化の妥協点としてツリーが良い表現であると考えている．抽象化という面では，ツリーはホスト集合を接続するネットワークグラフを少ないリンクの数で表現でき，そのまま階層的なクラスタリングとして用いることができる．またトポロジをツリーにモデル化することで，単純な手法で高速に推定することができる．一般化という面では，まず LAN 内のネットワーク，特にイーサネットのトポロジは通常ツリー構造である．通信路が冗長化された環境では，ツリーにモデル化することで通信性能の表現としては精度が低下する．しかし，推定対象とするクラスタの数がそれほど多くない場合は，実際に用いられる通信路が一定な環境も多い．

以降の構成を述べる．2 章で関連研究について述べ，既存の手法と我々の提案手法を比較する．3 章で測定が理想化された状況での推定アルゴリズムを紹介し，4 章で実環境で高速，高精度で推定するために重要な手法の詳細を述べる．そして 5 章で我々の推定手法を評価し，6 章で推定結果を用いたアプリケーションについて述べる．最後に 7 章で結論を述べる．

2. 関 連 研 究

ネットワークトポロジの推定に関する既存の研究はその目的によって，前提と手法が異なる．

インターネットのトポロジの発見に関する多くの研究では traceroute の推定結果を用いて推定を行う[5),7),15)]．また，SNMP などのプロトコルを用いてレイヤ 2 のトポロジを推定する研究もある[2),13)]．これらの研究の第 1 の目的はネットワークの管理やトラブルシュートであり，我々の目的とは大きく異なる．これらの研究の典型的な手法は非常に長い時間測定を続けてデータベースを構築する．一方我々の手法はネットワークへの負荷を抑え，短時間に計算に参加するホストだけのトポロジを推定する．

エンドホスト間の測定を用いる既存研究の多くは，測定結果を統計的に扱うことでトポロジを推定する[3),4),8)]．この手法はエンドホストで測定プロセスを起動するだけで推定を行えるため，多くの環境で適用できる．これらの手法の限界の 1 つは測定結果の変動によって推定結果も変動することである．そのため，管理プロトコルを用いた手法と比べて決定的な推定を行うことができない．もう 1 つの限界は，ホスト間の測定だけでは 1 本のリンクとスイッチで接続された 2 本のリンクの区別がつかないため，対象となるホストの分岐にかかわるスイッチしか推定できないことである．このようなトポロジを論理トポロジと呼び，物理トポロジとは区別される．ただし，論理トポロジでも複数の通信路でリンクを共有するかといった，並列アプリケーションの性能のために必要な情報の多くを有する．そのため並列アプリケーションの最適化のための情報という意味では，論理トポロジは物理トポロジの良い近似として扱うことができる．我々の手法もこれらの手法に分類される．

Duffield らの研究[8)] では，2 つの通信路の干渉の有無をパケットロス率の測定を用いて調べることでトポロジを推定する．しかしパケットロス率の測定には非常に長い時間がかかり，またネットワーク負荷が非常に高くなってしまう．Coates ら[4)] は 1.5 KB のパケットを 1 つのホストから他の推定対象のホストに送信することで推定を行う，ネットワーク負荷の低い手法を提案した．そして，アメリカにある 9 つのホストとポルトガルにある 2 つのホストの合わせて 11 ホストのトポロジを 8 分間かけて推定した．この手法は固定したホスト S から 2 つのホスト A と B に向けて 3 つのパケットを順に送信することで S–A 間と S–B 間の共通リンクのボトルネックになるバンド幅を測定する．具体的には，初めのパケットは A に，次のパケットは B に，最後のパケットは A に送信し，A に届いた最初と最後のパケットの遅延を測定する．この測定を全ホストペアについて行い，その結果により一致するトポロジを最尤推定を用いて推定する．この手法は測定回数が $O(N^2)$ になり，ホストの数が増加すると推定が非常に遅くなってしまう．また，Coates らの手法も含めて既存の推定手法の多くは，決められた 1 つのソースホストから他のホストへのパケットの送信だけを用いる．この方法はソースホストから遠い LAN 内のようなホスト集合のトポロジを推定することが困難である．我々の研究は Shao らの研究[14)] に近い．この研究は並列計算の最適化のために，エンドホスト間の測定だけを用いてトポロジを推定するフレームワークを提案した．この手法は 1 つのテストホストと他のそれぞれのホストとでバンド幅を測定し，同じようなバンド幅を持つホスト群にグループ分けをする．そしてグ

ループのうちの 2 つのホストとテストホストとでバンド幅を測定し，通信路の衝突の有無を調べる．衝突がなければ別のグループに分かれる．この手法は Shao らも論文中に述べているように，グループ内のバンド幅がテストホストとグループまでのバンド幅より大きい場合は，そのグループ内のトポロジについての情報を得られない．

我々の手法は，特に近いホスト間の RTT を重視することで LAN でも WAN でも高い精度の推定を期待できる．また遠くのホストとの必要のない測定を減らすことで短時間の推定を実現する．

3. RTT の測定に基づいたトポロジ推定

3.1 ネットワークトポロジモデル

我々の手法は，ネットワークをツリーにモデル化する．葉ノードは対象とするホスト（プロセス）を表し，中間ノードはスイッチを表す．またリンクには遅延を表す重みを与える．このアルゴリズムは論理トポロジを小さいメッセージの Round Trip Time（RTT）の測定を用いて推定する．この章では，測定される RTT がリンク単位で一定な理想的な状態での推定アルゴリズムについて述べる．測定するホストペアの選択方法や全プロセスでの動作手順，測定結果の変動の考慮といった，現実のネットワークで動作させるための詳細なアルゴリズムについては 4 章で述べる．

3.2 アルゴリズム

我々の推定アルゴリズムは，まず任意の 3 ホストを選んでツリーを推定し，そのツリーにホストを 1 つずつ加える．3 ホストのトポロジの形は**図 1** に一意に決まる．それぞれのリンクの重みは，各ホスト間の RTT の測定値を用いた以下の三元連立方程式の解である．AB，BC，CA はそれぞれのホスト間の RTT の測定値である．RTT はホスト間の遅延の 2 倍の値であるので，式の係数は 1/2 ではなく，1/4 になる．

$$\begin{aligned} x &= (\mathrm{AC}+\mathrm{AB}-\mathrm{BC})/4, \\ y &= (\mathrm{AB}+\mathrm{BC}-\mathrm{AC})/4, \\ z &= (\mathrm{BC}+\mathrm{AC}-\mathrm{AB})/4, \end{aligned} \tag{1}$$

次に，すでに推定されたツリーにホスト H を追加することを考える．推定されたツリー上の 2 ホスト A，B_0 を選択し，A，B_0，H からなる 3 ホストについて式 (1) を解くことで，推定されたツリー上での A，B_0，H の分岐点の位置を求める．この分岐点がとりうる位置は以下の 2 つに分かれる．

(1) **図 2** (a) のように推定されたツリー上で分岐点ではない場合，新たなスイッチ X を加え，H を X の先に加える．

(2) 図 2 (b) のように推定されたツリー上にすでに分岐 X がある場合，H が X をルートとするサブツリーのうち，A や B_0 を含まないもののいずれかに加わるということしか分からない．そこで，そのようなサブツリーに含まれる B_1 を選択し B_0（または A）と交代する．そして，A，B_1，H の 3 ホストについて同様の手順を行う．B_0 と A の選択については 4 章で述べる．

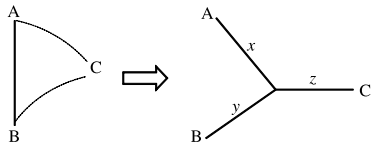
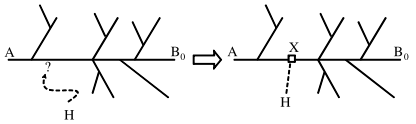
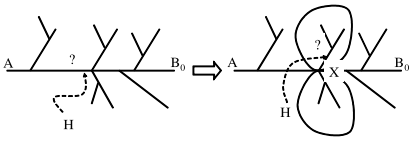


図 1 3 ホストのトポロジ
Fig. 1 The only topology of three hosts.



(a) A，B_0，H の分岐位置がツリーのリンク．



(b) A，B_0，H の分岐位置がツリーのスイッチ．H は丸で囲んだサブツリーのどちらかに含まれる．

図 2 2 つのホストと測定したときのとりうる追加位置
Fig. 2 Two possible outcomes of measuring RTTs to two hosts.

A，B，H の分岐位置 X を求めることで，H は A または B_0 を含むサブツリーの X 以外を分岐位置の探索空間から除くことができる．これを繰り返すことにより H が隣接する分岐位置が決まる．この手順（`add`）とトポロジ推定全体の手順（`infer`）を**図 3** に示す．

3.3 測定回数の上限

推定されたツリーを T，ホストの数を N，T 上の全ホスト間のホップ数の最大値（T の直径）を d，1 つのスイッチのポート数を p とする．このとき，手順 `add` の繰返しの回数が pd 以下になり (∗)，全体の手順 `infer` で $(pd+1)(N-2)$ 以下になる．このとき p が定数であるとすると，測定回数は $O(Nd)$ となる．

(∗) を示す．H を追加するときの繰返しで分岐位置 X は H から遠ざかることはなく，同じ X となるか，実際の H の位置に近づくかのどちらかである．1 回の繰返しで少なくとも 1 つのサブツリーを除くことがで

```
add(T, H) : /* add host H to tree T */
  M = all hosts (leaf nodes) of T;
  A = choose and remove an arbitrary host from M;
  measure AH (RTT between A and H);
  while (M is not empty) :
    B = choose and remove an arbitrary host from M;
    measure BH;
    X = find the branching point
          of A, B, and H by (1);
    if (X does not coincide with
       an existing branching point) :
       add X to T as a new branching point;
    /* this line has effect only in the first iteration */
    remove from M all hosts under
          the subtree rooted at X containing A;
    remove from M all hosts under
          the subtree rooted at X containing B;
    A = either A or B; /* arbitrarily (see the text) */
  connect X and H;

infer(H) :
  /* H is a list of hosts H_0, H_1, ... */
  T = the tree containing only H_0 and H_1;
  for H_i (i = 2, 3, ...) :
    add(T, H_i);
  return T;
```

図 3 基本的なトポロジ推定アルゴリズム
Fig. 3 Basic topology inference algorithm.

きるので，X にとどまる回数は最大で p 回である．したがって，add の繰返し回数は pd 以下になる．d は通常（$O(\log N)$）より小さく，このアルゴリズムは高いスケーラビリティを持つ．

4. 実際のネットワークにおける測定と推定

実環境では測定結果の変動は避けられず，それによって推定結果も誤ったものになる．この章では実環境での運用時において重要な技法について述べる．

4.1 一対のホスト間の測定

一対のホストペアの RTT の測定は，TCP パケットの ping-pong を複数回繰り返し，送信から受信までの時間を用いる．送信ホストは測定された RTT の最小値を記録し，10 回連続で最小値を更新することがないか，決められた上限の回数繰り返したら測定を終了する．この上限を今は 30 回としている．この手順を 3 回繰り返し，各セットでの最小値を 3 つ得て，その最大値と最小値の差を分岐の区間とする．手順 add 内の繰返しでは，その区間の中心に最も近い分岐をトポロジ上の分岐 X とする．区間を用いることで，測定結果のわずかな変動によって小さい遅延のリンクを持つスイッチの発生を抑え，同じスイッチにつながるホストやスイッチの分離を防ぐ．

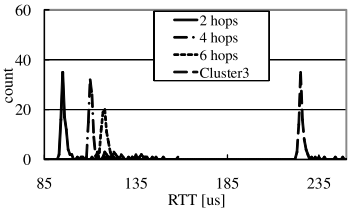


図 4 Cluster 4 内および 3 との RTT の測定値の分布
Fig. 4 Distribution of RTTs in Cluster 4 and to Cluster 3.

ここまでで 3 つのパラメータを示したが，測定の繰返しはこの程度の回数で安定した値を得られると考えている．**図 4** は 1 つのクラスタ内で 2，4，6 ホップ離れたホスト（2，4，6 hops）間，および 3 つのレイヤ 3 ルータをはさんだ別のクラスタのホスト（Cluster 3）間で 100 回測定した RTT の分布を示す．クラスタ内では 2 ホップにつき RTT が $10\,\mu$ 秒ほど増加し，同一ホストペアの測定の変動より大きい．このような分布であれば，先ほど述べたサンプリング手法によって特にホスト H と近いホストとより離れたホストを区別することができる．

4.2 測定するホストの選択

このアルゴリズムを，推定中のツリーをホストの間で転送させてツリーに自ホストを追加するという方針で実装する．ホストを $\mathrm{H}_0, \mathrm{H}_1, \ldots$ と並べて，ホスト H_i は $\mathrm{H}_0, \ldots, \mathrm{H}_{i-1}$ を含むツリーが到着するのを待つ．到着したら H_i を加えたツリーを H_{i+1} に送る．この順序は対象ホスト群をランダム順に並べて構成する．以降では H_i を加えたツリーを T_i と表現する．

測定の変動による影響をできるだけ小さくするためには，近いホスト間の測定結果を用いることが重要である．近いホスト間の測定結果は比較的分散が小さく，また誤った位置に追加したとしても誤りの距離が小さくなる．さらにこれには手順 add での探索空間を大きく削減し，繰返し回数を減らすという効果もある．これらを実現するために，以下の 2 つを行う．

- 手順 add のイテレーションの最後で A，B_1 ともに H に近いホストを選択する．
- add のイテレーションの最初の A を選択するときに H にできるだけ近いホストを選択する．10 ホスト前からツリーを受け取り，他のホストの追加と並行して選択を行う．

前者は分岐位置と推定中のツリーに含まれるホスト

との距離が必要であるが，これは推定中のツリーを用いて求めることができる．一方後者は，追加しようとするホストと構築中のツリーに含まれるホストとの距離が必要であり，追加の前に測定をする必要がある．この測定を多数のホストと行うと時間的コストが大きくなってしまう．そこで，以下のように測定を進めることで，遠いホストとの測定を削減する．H_i は T_i を構築した後 H_{i+1} に T_i を送り，同時に H_{i+10} にも同じ T_i を送る．H_{i+10} は T_i を受け取ってすぐに T_i に含まれるホストのなかで近いホストの探索を開始する．i が十分に大きければ T_i 中の近いホストと H_{i+10} が後に受け取るべき T_{i+9} 中の近いホストはほぼ同じになる．測定候補のホストは T_i に含まれる全ホストであるが，以下のような基準によって測定するホストを制限する．ホスト H が測定候補から無作為に選択したホスト J との RTT の測定値 x を得たとき，測定候補に残っているホスト K について，T_i 上の J と K のパスの重みの和が $3x$ より大きいかまたは $x/3$ より小さいときに候補から外す．これは，H と K の RTT が前者では $2x$ より大きく，後者は $2/3x$ より大きくなり，直感的には K が J より十分遠いか，または J と同程度に近いことを意味する．この制限は特にマルチクラスタ環境で効果的である．クラスタ内の RTT がクラスタ間の RTT より十分小さい（1/3 以下）とき，H は他のクラスタのホストとせいぜい 1 つずつしか測定しない．この手順を**図 5** に示す．

```
find_close_host(T′) :
  M = all hosts in T′;
  while (M is not empty) ;
    choose and remove an arbitrary host J from M;
    x = RTT between H and J and record it;
    for all K ∈ M :
      y = distance between K and J on T′;
      if (y > 3x or y < x/3) remove K from M;
  return the host that had
    the minimum RTT in the loop;
```

図 5 近いホストの選択の手順
Fig. 5 The procedure of find close host.

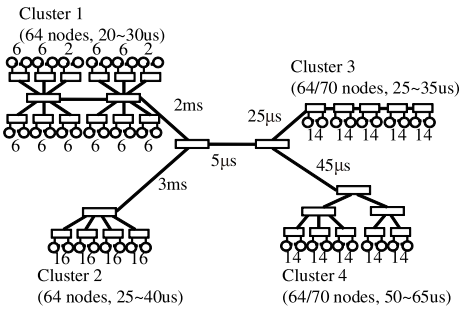


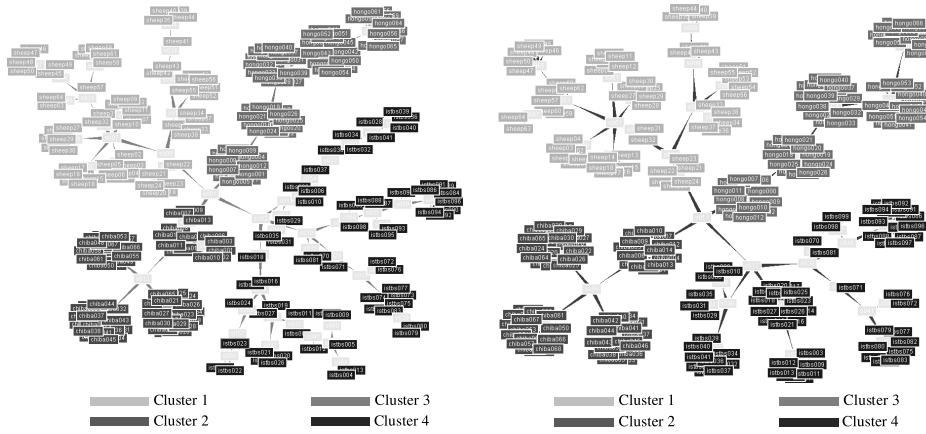
図 6 実験環境
Fig. 6 The experimental environment.

5. 評　　価

我々の手法を用いて**図 6** のような 4 クラスタ 256 ホストの環境で実験を行った．各クラスタはレイヤ 2 スイッチで構成されており，クラスタ間はレイヤ 3 ルータ構成されている．図中の時間はクラスタ間およびクラスタ内の遅延を示す．クラスタ内ではホップ数によって異なる遅延が測定される．すべてのホストの OS は Linux で，ネットワークインタフェースは Gigabit Ethernet である．Cluster 3 と 4 は同じ建物に，Cluster 1 は 25 km ほど，Cluster 2 は 31 km ほど離れており，クラスタ間の遅延はそのペアによって異なる．

5.1 推 定 精 度

まず，我々の手法を用いて推定したツリーを**図 7** (a) に示す．同じ色のホストは同じクラスタに属すことを意味し，この図から同じクラスタのホストを識別できたことが分かる．さらにクラスタ内の同じレイヤ 2 スイッチに接続されるホストを識別することもでき，その他のスイッチについても部分的には推定できた．たとえば左下の Cluster 2 のホスト群は十字に分かれ，右上の Cluster 3 のホスト群は一列に並んでいる．この環境で多く発生する誤りは主に 2 種類であった．1 つ目は左上の Cluster 1 や右下の Cluster 4 のように同じスイッチにつながるホストが複数のスイッチに分かれることである．図 7 (b) はこの誤りだけを手動で訂正したものである．具体的には，隣接する 2 つのスイッチにつながるすべてのホストが実際のトポロジ上で同じスイッチにつながるものであったときだけその 2 つのスイッチをマージするという処理を行った．図 7 (b) では Cluster 1 や 4 でも同じスイッチにつながるホスト群がグループ分けされている．つまり，本来の推定結果の図 7 (a) でも同じスイッチにつながるホスト群が近い位置に推定されていることが分かる．このような分離が生じた理由は，4.1 節で述べた推定値の区間が Cluster 4 では小さすぎたためである．上記の処理でマージされたスイッチ間のリンクの遅延は残ったリンクの遅延よりも小さかった．スイッチ間をマージするリンクの重みのしきい値を適応的に定めることが課題として残る．2 つ目は，小さな遅延のリンクの近くにホストがない場合に小さな遅延のリンクとその両端のスイッチが 1 つのスイッチと推定されるこ



(a) 推定結果 (b) 分離したスイッチのマージ処理を行った結果

図 7 4 クラスタでの推定結果のトポロジ
Fig. 7 An inferred topology of 4 clusters.

とである．これは遠いホスト間の測定を用いてホストを追加する際の測定に，そのリンクの遅延以上の分散が生じることが原因である．図 6 の環境では，Cluster 1，2 と Cluster 3，4 の間の 5 μs のリンクとその両端のスイッチが図 7 では隠れて 1 つのスイッチと推定されることが多かった．この問題については，RTT だけを用いる手法では解決が困難であり，他の測定を併用することを考える必要がある．

推定されたトポロジの精度を定量的に評価する指標として “A と B，C と D の通信路が共有するリンクの有無” の問合せの正誤を用いる．我々の手法で得られるトポロジは確率的であるので，ここでは 100 通りの推定結果について各推定結果での誤り率を求め，その分布を示す．1 つの推定結果に対して全ペアを調べることは計算量が膨大で不可能であるので，ここでは 2 つのホストペアを無作為に 10 万回選択し，実際のトポロジでのリンクの有無とは異なった答えを返した回数を求めてその割合を誤り率とした．Cluster 4 と全クラスタでの誤り率の分布を**図 8** に示す．左の図は，推定結果から共有リンクがないと回答されたが実際は共有リンクがある場合の誤り（false negative）を，右の図はその逆（false positive）を示す．単一のクラスタでは先に述べたスイッチの分離によって false positive が高いが，一方で false negative は非常に小さかった．4 クラスタでは false negative が高くなった．これは，図 6 の 5 μs のリンクを推定できず，1 つのスイッチに 4 つのクラスタがつながるように推定されることが多かったためである．この場合，Cluster 1 から 3 と 2 から 4 のように 5 μs のリンクを共有する通信パターンで共有するリンクがないと判定してしまう．

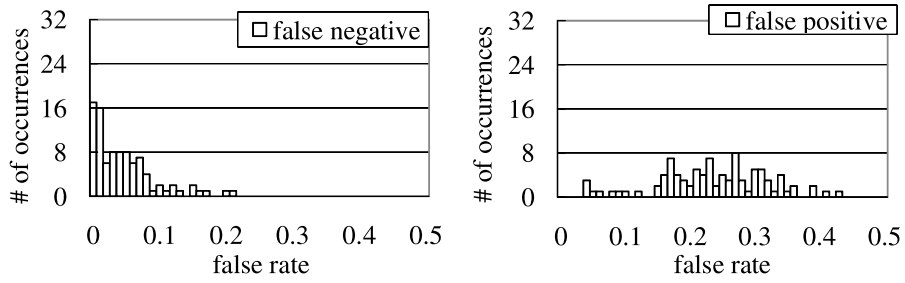
5.2 推 定 時 間

クラスタ数，ホスト数ごとの推定時間を**図 9** に示す．また比較対象として，4.2 節で述べたような測定するホストの制限を行わずに全対全の測定を行う場合の 4 クラスタでの推定時間を all-to-all として示す．複数クラスタでは各クラスタで同じ数のホストを用いた．我々の手法は推定時間は 4 クラスタ 256 ホストで 15 秒程度であり．このグラフから高いスケーラビリティを有することが分かる．一方，all-to-all は 128 ホストまでは我々の手法と同程度であったが，それ以上ではホストが増えるにつれて我々の手法よりも推定時間が増加した．このことから，測定ホストを制限することでスケーラビリティが向上することが分かる．クラスタ間またはクラスタ内で実際に測定を行った回数を**図 10** に示す．同じクラスタのホストとは全体 $64 \times 63/2$ ペアのうち 35%から 60%と測定したが，他のホストとは全体 $64 \times 64/2$ ペアのうちの 2%程度としか測定しなかった．すなわち，遠いホストとの測定を制限することができたということが分かる．

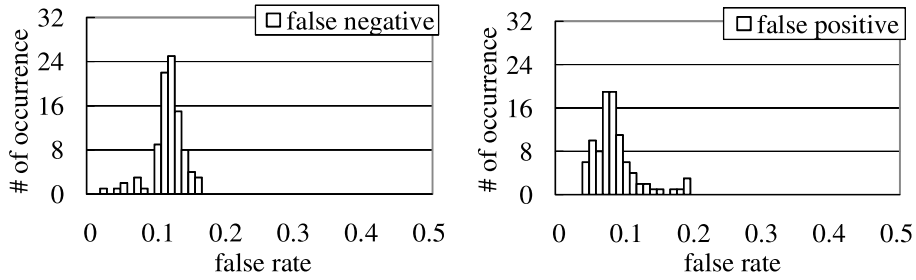
6. 推定結果を用いたアプリケーション

6.1 バンド幅マップの構築

枝の重みにバンド幅を与えたツリーをバンド幅マップと呼ぶ．このバンド幅マップを用いることで単一のホスト間のバンド幅や複数ホストペアで同時に通信したときに得られる実効のバンド幅も求めることができる．本章ではツリーを用いてバンド幅マップを構築する手法について述べ，推定したツリーを用いて構築したバンド幅マップから得られるバンド幅情報の精度を



(a) a single cluster (cluster 4)



(b) four clusters

図 8 共有リンクの問合せについての誤り率の分布
Fig. 8 Distributions of false rates to queries on shared links.

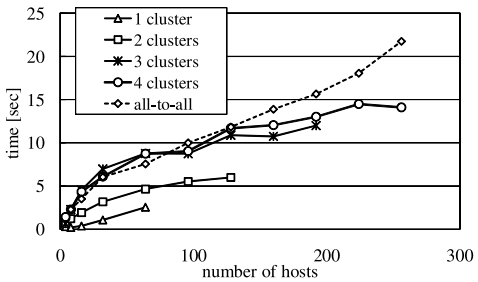


図 9 推定時間
Fig. 9 Inference time.

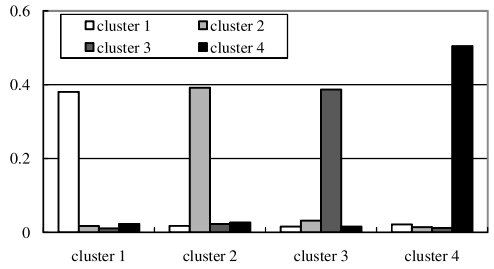


図 10 測定されたホストペアの割合
Fig. 10 Percentages of pairs measured.

示す．バンド幅を並列に測定する場合，同時に測定をするホストペアがボトルネックリンクを共有するとお互いが測定の邪魔をし，正確なバンド幅を測定できない．したがってトポロジ情報を用いない場合，バンド幅の測定を逐次に行わなければならない．我々の手法によって推定されたツリーを用いることで，バンド幅マップの構築のために測定が必要なホストペアと，その測定の効率的なスケジュールを得られる．

バンド幅の測定をいくつかのステージに分け，各ステージではリンクを共有しないと分かるホストペアでのみ測定する．まず推定されたツリー上の任意のスイッチを 1 つ選び，そのスイッチをツリーのルートとする．ルートスイッチは深さ 0 とし，スイッチやホストをそれぞれの深さ（ルートからのホップ数）で分類する．ステージ i ($i = 1, 2, \ldots, h$，h はツリーの高さ）では深さ $(h - i)$ にあるスイッチとその子ノード（スイッチまたはホスト）との間のリンクのバンド幅を測定する．子ノードがスイッチの場合は，そのスイッチをルートとするサブツリーに含まれるホストのうち，そのスイッチまでのバンド幅が最も大きいと期待されるホストを用いる．この手法では深いリンクのバンド幅を先に測定するので，サブツリーのホストとスイッチの間のバンド幅は構築中のバンド幅マップから求められる．

深さ $(h - i)$ にあるスイッチ X が p 個の子ノード $\mathrm{C}_1, \ldots, \mathrm{C}_p$ を持ち，H_i を C_i のサブツリーから選択されたホストとする．また，XC_i をスイッチ X とその子ノード C_i の間のリンクとする．ここではすべての i について，XC_i のバンド幅を求めることが目的である．$\{\mathrm{H}_i\}$ を $p/2$ 個のペアに分け，それぞれのホスト間で同時に測定を行い，H_i と H_j との測定によっ

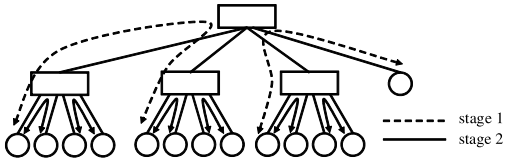


図 11 バンド幅マップ構築のスケジューリング例
Fig. 11 An example of schedule for building bandwidth map.

て得られたバンド幅を XC_i 間のリンクと XC_j 間のリンクのバンド幅とする．もし p が奇数であれば，先ほどの測定が終わった後で余ったホストと，X とのバンド幅が最も大きいと期待されるホストとで測定を行う．そのため 1 ステージで 1 つのホストは最大 2 回測定を行う．深さ 2 のツリーで以上の処理を行うと，測定するホストペアは**図 11** のようになる．stage1 − 1, 1 − 2 は同じステージでの 1 度目の測定と 2 度目の測定を意味する．最後の手順は，深いリンクのバンド幅が小さいときに浅いリンクのバンド幅が実際より小さくなることがある．しかし大規模なマルチクラスタ環境で非常に効率良く妥当な結果を得られることから，この手法は現実的な環境で有用である．この手法はクラスタ内で実際に得られるバンド幅を求めることができ，さらにクラスタ間のリンクのボトルネックを見つけることができる．また我々は現在，より一般的な環境においてバンド幅マップを構築する手法についても研究している．

この手法は深さに比例した時間でバンド幅マップを構築できる．ホスト数を N としたときツリーの深さが $O(\log N)$ であり，高いスケーラビリティを得られる．この手法を用いてバンド幅マップの構築にかかった時間を**図 12** に示す．4 クラスタ 256 ホストの環境で 30 秒程度でバンド幅マップを構築できた．また 96 ホスト以上では構築時間はそれほど増えなかった．

バンド幅マップの精度の定量的な評価として，ホスト間の通信路の最小のバンド幅と iperf で測定したバンド幅の比を**図 13** に示す．ただし，遅延が大きいホスト間では TCP のウィンドウサイズによる制限のためにホスト間のバンド幅が物理的な空き帯域よりも小さくなる．そこで，得られるバンド幅はバンド幅マップによって得られるホスト間のバンド幅と，ウィンドウサイズと遅延によって求まるバンド幅のうちの小さい方の値を用いた．1 クラスタでは多くのホスト間で正しく測定でき，20%ほどのリンクで衝突が生じた．4 クラスタでは実際のバンド幅よりツリーから得られたバンド幅が大きくなることがあったが，これも 5.1 節と同様に Cluster 1，2 と Cluster 3，4 の間の 5 μs の

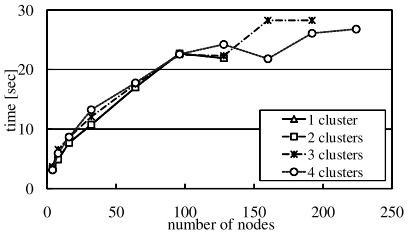
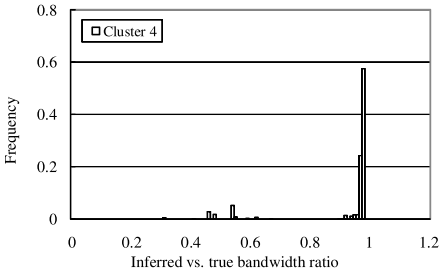
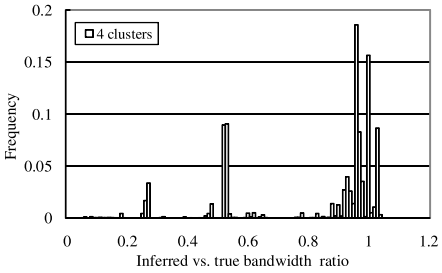


図 12 バンド幅マップの構築にかかる時間
Fig. 12 Time to build bandwidth map.



(a) 1 クラスタ



(b) 4 クラスタ

図 13 バンド幅マップの精度
Fig. 13 Accuracy of bandwidth maps.

リンクを推定できないことが原因である．図 7 (a) では Cluster 1 と 3 の間と 2 と 4 の間で測定が行われ，約 200 Mbps という測定結果を得て Cluster 1 と Cluster 2 の間のバンド幅を 200 Mbps と回答する．しかし Cluster1-2 間は遅延が 2 + 3 ms 程度と大きく，直接の測定では 150 Mbps 程度しか得られなかった．ウィンドウサイズと遅延によって求まる値は 200 Mbps であり，直接測定して得られるバンド幅よりも大きかった．

6.2 長いメッセージのブロードキャスト

この節では，IP ユニキャストだけを用いて長いメッセージを全ホストにブロードキャストするための効率良い転送パターンを，トポロジを用いて導く手法を紹介する．実際のトポロジがツリー構造のときは，各リンクを片方向 1 度ずつ通るようにホストを並べてメッセージをパイプライン転送することで最大のバンド幅を得られる．我々は推定したトポロジ上の深さ優先順に並べることでそのようなパイプラインを構成した．

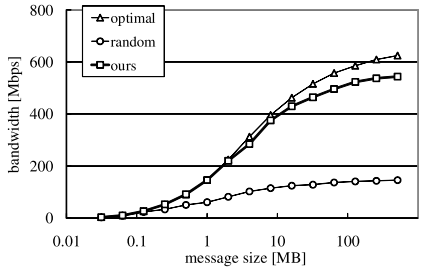


図 14 クラスタ 4 内でのブロードキャストで得られたバンド幅
Fig. 14 Obtained bandwidth at broadcast in cluster 4.

実際のトポロジがツリーでないときは，文献 6) のように冗長化されたパスを用いてより高いバンド幅を得ることができるが，特にイーサネットで構成された LAN では多くの場合トポロジはツリー構造であり，またクラスタ数がそれほど多くなければ WAN であっても通信路が一定でツリーをなすことが多い．そのような環境では我々の手法が効果的である．

Cluster 4 内でブロードキャストを行い，得られたバンド幅を測定した．スイッチ間は 4 本のリンクによるトランクをなす．我々の手法（ours）との比較対象として，実際のトポロジ上で深さ優先順に並べた最適なパイプラインを用いる手法（optimal）と，ランダム順に並べたパイプラインを用いる手法（random）についても測定を行った．**図 14** に 3 つの異なるアルゴリズムの結果を示す．optimal は 624 Mbps のバンド幅が得られ，ours はその 88%の性能を得られた．我々の手法はいくつかのリンクを 2 度使用するが，スイッチ間のトランクのためにそれほど大きな影響を受けない．しかし，random はスイッチ間のリンクを多数回使用し，optimal の 23%の性能しか得られなかった．

7. おわりに

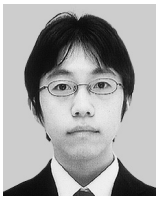
本稿では効率的なトポロジ推定アルゴリズムと推定結果を用いたアプリケーションについて述べた．実験の結果 1 クラスタ 64 ホストの推定は 4 秒程度，4 クラスタ 256 ホストの推定は 15 秒程度であり，我々の推定アルゴリズムは高いスケーラビリティを持つことが示された．推定結果はクラスタの判別はもちろん，クラスタ内の同じスイッチにつながるホストのグループ分けを確認できた．今後はまず，スイッチ間の遅延のしきい値を決定するより良い方法を考案し，推定精度の向上を目指す．そして遠いホストどうしは独立に追加できるということを考えて，複数ホストを並列に追加することによる高速化を目指す．また，バンド幅マップの構築における現在の課題を解決し，より一般的なトポロジへの適応を目指す．

参考文献

1) Aumage, O. and Mercier, G.: MPICH/MAD III: A Cluster of Clusters Enabled MPI Implementation, *3rd International Symposium on Cluster Computing and the Grid*, pp.26–33 (2003).
2) Breitbart, Y., Garofalakis, M., Jai, B., Martin, C., Rastogi, R. and Silberschatz, A.: Topology discovery in heterogeneous IP networks: The NetInventory system, *IEEE/ACM Trans. Netw.*, Vol.12, No.3, pp.401–414 (2004).
3) Byers, J.W., Bstavros, A. and Harfoush, K.A.: Inference and Labeling of Metric-Induced Network Topologies, *IEEE Trans. Parallel Distrib. Syst.*, Vol.16, No.11, pp.1053–1065 (2005).
4) Coates, M., Castro, R., Nowak, R., Gadhiok, M., King, R. and Tsang, Y.: Maximum likelihood network topology identification from edge-based unicast measurements. *SIGMETRICS Perform. Eval. Rev.*, Vol.30, No.1, pp.11–20 (2002).
5) den Burger, M., Kielmann, T. and Bal, H.E.: "TOPOMON": A Monitoring Tool for Grid Network Topology, *ICCS '02: Proc. International Conference on Computational Science-Part II*, pp.558–567 (2002).
6) den Burger, M., Kielmann, T. and Bal, H.E.: Balanced Multicasting: High-throughput Communication for Grid Applications, *Proc. 2005 ACM/IEEE Conference on Supercomputing* (2005).
7) Donnet, B., Raoult, P., Friedman, T. and Crovella, M.: Efficient algorithms for large-scale topology discovery, *SIGMETRICS Perform. Eval. Rev.*, Vol.33, No.1, pp.327–338 (2005).
8) Duffield, N.G., Horowitz, J., Presti, F.L. and Towsley, D.: Multicast topology inference from measured end-to-end loss, *IEEE Trans. Inf. Theory*, Vol.48, pp.26–45 (2002).
9) Gabriel, E., Resch, M., Beisel, T. and Keller, R.: Distributed Computing in a Heterogeneous Computing Environment, *Proc. 5th European PVM/MPI Users' Group Meeting*, pp.180–187 (1998).
10) Imamura, T., Tsujita, Y., Koide, H. and Takemiya, H.: An Architecture of Stampi: MPI Library on a Cluster of Parallel Computers, *Proc. 7th European PVM/MPI Users' Group Meeting*, pp.200–207 (2000).
11) Karonis, N.T., Toonen, B. and Foster, I.: MPICH-G2: A Grid-enabled implementation of the Message Passing Interface, *J. Parallel Dis-*

trib. Comput., Vol.63, No.5, pp.551–563 (2003).
12) 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, *PPoPP '99: Proc. 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp.131–140. ACM Press (1999).
13) Lowekamp, B., O'Hallaron, D. and Gross, T.: Topology discovery for large ethernet networks, *SIGCOMM '01: Proc. 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp.237–248. ACM Press (2001).
14) Shao, G., Berman, F. and Wolski, R.: Using Effective Network Views to Promote Distributed Application Performance, *Proc. 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, pp.2649–2656 (1999).
15) Skitter. http://www.caida.org/tools/measurement/skitter

(平成 19 年 1 月 22 日受付)
(平成 19 年 4 月 26 日採録)



白井 達也（正会員）
1983 年生．2007 年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修了．同年より株式会社リコー勤務．高速計算の基盤技術に興味を持つ．IEEE 会員．

斎藤 秀雄（学生会員）
1981 年生．2006 年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修了．同年より同博士課程在学中．ACM 会員．

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）．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講師．2002 年より同助教授．2007 年より同准教授．日本ソフトウェア科学会，ACM，IEEE-CS 各会員．