修　　士　　論　　文

高速なトポロジー推定 — ネットワークを考慮した並列計算の基盤として

A Fast Topology Inference — A building block for network-aware parallel processing

指導教員　**田浦 健次朗 助教授**

東京大学情報理工学系研究科
電子情報学専攻

氏 名　**48-56424 白井 達也**

提 出 日　**平成 19 年 2 月 2 日**

概 要

頻繁に通信を行う並列アプリケーションの性能向上にとって，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．

本論文ではより汎用的な情報としてネットワークトポロジー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．本研究の手法は 1 クラスタ 64 ホストのトポロジーを 4 秒程度で，4 クラスタ 256 ホストのトポロジーを 15 秒程度で推定し高いスケーラビリティを得られた．さらに推定したトポロジーを用いてバンド幅ツリー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．

目 次

図 目 次

第1章　序論

1.1　背景と目的

通信を頻繁に行うアプリケーションや多くの集合通信の性能は，物理ネットワーク上の通信路や通信パターンによって大きく変動する．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．例えば，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非常に重要になる [14]．また，ネットワークの性能がほぼ均一な LAN 内であっても，複数の通信がひとつのリンクを用いると得られるバンド幅が低下してしまう．

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．例えばグリッド環境で動作する MPI に関する多くの研究 [1, 10, 12, 13] では，MPI のシステムが用いるホスト間の局所性が分かっており，グループ分けが与えられているということを仮定する．このグループ分けを手動で設定することようなシステムは，全てのユーザが同じホスト集合のみを用いて，さらに使えるホスト集合が滅多に変更されないような静的な環境でしか用いることができない．管理者が異なる複数のホスト群（例えば異なる管理者が管理する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環境では，それぞれのユーザに合わせた設定を記述する必要がある．また使えるホスト群が頻繁に変更される，よりオープンで動的な環境では設定の更新が大きな手間になる．さらに，ソフトウェア的にヘテロなホスト群で仮想マシンを構築し，それらを VLAN で接続するといったような仮想化された環境 [11] では，IP アドレスやホスト名はネットワーク的にほとんど意味を持たず，設定を行うこと自体が困難である．また，既存のシステムの多くはホスト集合をクラスタ間とクラスタ内の 2 つに分けるだけである．しかし現実には単一のクラスタ内でも複数のスイッチで構成されている場合は，クラスタ間と同様にスイッチ間の通信を減らす必要が生じる．さらに，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できない．システムが得たネットワーク的な情報がプログラマに提供されないため，プログラマが実現したい通信パターンの最適化には用いられない．

ネットワークを考慮したアプリケーションを記述するためには，まずネットワークの情報を得る必要がある．しかし上述したように，ネットワーク情報を手動で記述することは現実的ではなく，何らかの測定によって動的に情報を得て，それをアプリケーションの通信性能の向上に有用な表現を用いて提供する必要がある．そのような表現として，ネットワークトポロジーのツリーモデル表現が提案された [17]．ネットワークトポロジーを手動の設定を用いずに推定する手法に関する研究はこ

れまで数多くなされてきた．これまでの手法は長時間測定を続けて統計的にトポロジーを推定するものが多い．しかし先に述べたような動的な環境ではユーザごとに用いるホスト群が異なり，用いるホスト群のトポロジーを前もって用意することができない．このような環境でネットワークの性能情報を有するトポロジーを推定するためには，計算開始直前に短時間に推定する必要がある．

1.2 本研究の貢献

本論文ではネットワークトポロジーのツリーモデルを，ホスト間の RTT（小さいパケットのホスト間の通信往復時間）の測定だけを用いて推定する手法について述べる．我々が提案する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，手動の設定を一切必要とせずにネットワーク的にヘテロな環境で動作する．また本論文では推定したトポロジーを用いてネットワークを考慮したアプリケーションを実装し，推定したトポロジーが通信性能の向上に役立つことを示す．推定アルゴリズムに必要な情報は推定対象のエンドホストの IP アドレスとポートだけであり，それらを接続するスイッチの情報は必要ない．推定されたツリーはリンクの重みとして遅延の情報を有する．

我々が提案する推定手法は RTT の測定結果だけを用いる．対象ホストを接続するスイッチは推定によって導き出される．RTT だけを用いる手法には大きく 2 つの利点がある．まず，traceroute や SNMP などの特別なプロトコルを用いないことである．これらのプロトコルによる情報はセキュリティポリシーや，そもそもハードウェアが対応していないなどの理由により，特に外部から得られないことがある．そして RTT の測定はネットワークへの負荷が小さく，他のプロセスとの干渉が非常に小さいことである．

推定対象のホストの数を N，ネットワークの直径を d としたとき，我々のアルゴリズムは理想的な状況下では $O(Nd)$ の測定回数で推定することができる．そして実装の工夫により 1 クラスタ 64 ノードのトポロジーを 4 秒程度で，4 クラスタ 256 ノードのトポロジーを 15 秒程度で推定した．その推定結果は個別のクラスタの判別はもちろん，クラスタ内のグループ分けも部分的に成功し，我々の方法は LAN 内，LAN 外の区別なく適用できた．

また推定したトポロジーを用いて（1）バンド幅ツリーの構築のための測定の効果的な並列化,（2）大きいメッセージの最適化，そして（3）MPI アプリケーションの Rank 割り当てを実現した．

1.3 本論文の構成

第 2 章 関連研究 トポロジー推定に関する既存の研究を紹介し，それらにたいする本研究の位置付けを述べる．

第 3 章 RTT の測定にもとづいたトポロジー推定 本研究の基本となるアルゴリズムとして，測定結果が変動しない理想的な状態での推定手法を述べる．

第 4 章 実環境における測定と推定手法 測定結果が変動する実環境でそれらの影響を減らし，短時間で推定する手法について述べる。

第 5 章 実験と評価 本手法を実環境で実験した結果を述べる。

第 6 章 トポロジーを用いた通信最適化 トポロジーを用いた通信の最適化手法の例をいくつか挙げる。そして、本手法で推定したトポロジーを用いて評価を行い、推定したトポロジー情報を用いた通信最適化という方向性が現実的であることを示す。

第 7 章 結論 本論文の寄与と今後の課題について述べる.

第2章　関連研究

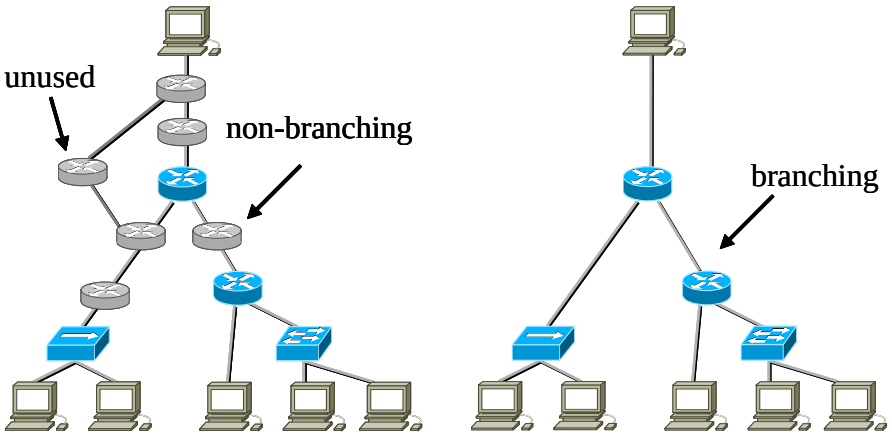
ネットワークトポロジーの推定に関する既存の研究はいくつかの目的に分かれる．その目的によって，前提と手法が異なる．この章ではまず，トポロジーの形態として物理トポロジーと論理トポロジーを紹介する．そして，それらのトポロジーを推定する既存の手法について述べ，我々の手法との比較を行う．最後に，ネットワークトポロジーを用いたグリッドアプリケーションの研究について述べる．

2.1　物理トポロジーと論理トポロジー

この節では物理トポロジーと論理トポロジーの区別について述べる．物理トポロジーとは，図2.1(a)のようにネットワークそのものを表現したトポロジーであり，対象とするネットワークに存在するすべてのホストとスイッチ，ルータなどのハードウェアを含む．ネットワーク管理やトラブルシュートで，接続機器そのものの存在が重要なときに物理トポロジーが重要になる．それに対して論理トポロジーとは，図2.1のように対象とするホスト群の接続形態のことであり，そのホスト群の分岐に関わらないスイッチは除かれる．並列計算では対象とするホスト群の分岐の関係と，ホスト間の通信性能の情報が重要であり，ひとつのリンクであるのか分岐に関係ないスイッチをはさんだ2つのリンクであるのかの区別は特に重要ではない．したがって，我々の推定システムは論理トポロジーを推定することができればよいと考える．

2.2　物理トポロジーの推定手法

大規模なインターネットの物理トポロジー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 [6, 18, 8]．これらの研究の第一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できるだけ多く，何百万ものホストやルータを発見し，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ーを推定する．また，我々はレイヤ3のルータだけでなくレイヤ2のスイッチを推定することが目的であり，またそれが可能である．SMNP のようなプロトコルを用いてレイヤ2の物理トポロジーを推定する研究もある [3, 15] が，スイッチなどの接続機器から情報を得るための設定ができる権限を持つものは管理



(a) 物理トポロジー (b) 論理トポロジー

図 2.1: トポロジー表現

者に限られており，ユーザアカウントしか持たないような一般ユーザだけでは動作させられないことが欠点としてあげられる．これらと異なる手法として，Black ら [2] はイーサネットに限られたものであったが，レイヤ 2 のトポロジーをエンドホストによるパケットの監視によって推定する手法を示した．しかし，この手法も動作環境が限られており，管理ポリシーによっては動作しないと考えられる．

2.3 ホスト間の測定を用いた論理トポロジーの推定手法

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ーを推定する [9, 4, 5]．我々の手法もこの手法に分類される．この手法は管理者権限が必要な情報や管理ポリシーによって得られないような情報を用いることなく，エンドホストが通信プロセスを起動できれば推定を行えるため，多くの環境で用いることができる．この手法のその他の利点としては，トポロジーだけでなく遅延やバンド幅といったネットワークの性能も同時に得られることがあげられる．また，ホストの増減が激しい動的な環境にも適応できる．一方で我々の手法を含めたこれらの手法の限界の一つは，測定結果の変動によって推定結果も変動してしまうことである．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．もう一つの限界は，論理トポロジーしか推定できず，特に多くの研究ではツリートポロジーを仮定していることである，例えば遅延を測定して推定する場合に，スイッチやルータをはさんだ 2 本のリンクと，一本のリンクとは原理的に区別できない．しかし，論理トポロジーは複数の通信路が一つの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は有する．ネットワーク管理やトラブル

シュートいった目的の場合は論理トポロジーでは不十分だが，並列アプリケーションの最適化という目的の場合は論理トポロジーは物理トポロジーの抽象化として扱うことができる．

Duffield ら [9] は 2 つの通信路が共有リンクを持つかどうかをパケットロス率の測定を用いておこなうことでトポロジーを推定した．しかしこの手法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 [5] は 1.5kB 程度のパケットを一つのホストから他の推定対象のホストに向かって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，8 分間かけてアメリカにある 9 つのホストとポルトガルにある 2 つのホストのトポロジーを推定した．この手法は定まったホスト 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ー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．

これらの推定手法は共通して，決められた一つのホストから他のホストへのパケットの送信だけを用いている．しかしこの方法は決められたホストから遠いホストの集合（e.g. LAN のホスト集合）のトポロジーを推定することが困難である．例えば Coates らの手法は，ホスト S のすぐ先のリンクのバンド幅よりも大きいバンド幅をもつ LAN 内については均一なバンド幅が測定される．このような場合 LAN 内のトポロジーの情報を得ることができない．したがって，この手法はそれぞれのホストが疎に存在しているときしか用いることができない．一方，我々の手法は全ての推定対象のホストが必要に応じて測定パケットを送信することで，特に近いホスト間の測定を用いてトポロジーを推定し，また遠いホスト間の測定を減らす．

2.4 ネットワークを考慮したグリッドアプリケーション

ネットワークを考慮した並列アプリケーションのフレームワークに関する研究について述べる．Network Weather Service[20] は実際に得られるバンド幅などのネットワークの状態を提供するシステムとしてよく知られている．ただし，NWS は近い将来に得られる性能を推測することを目的としており，トポロジー推定は行わない．我々の総合的な提案は Shao らの研究 [17] 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ーを推定するフレームワークを提案した．この手法は一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もつホスト群にグループ分けする．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，通信の衝突の有無を調べる．もし衝突がなければそれらはべつのグループにわかれるということが分かる．この手法は Shao らも論文中に述べているように，Coates らの手法と同様の問題，すなわち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ーについての情報を得られない．

金田らは RTT とバンド幅の測定によってクラスタ内の 112 ノード，10 スイッチからなるトポロジーを数時間かけて高い精度で推定した [21]．そして，それをもとに並列アプリケーションにおけるホスト間の接続の数の削減を実現した．この手法は，まず RTT を用いてノードと直接接続されたスイッチを推定してホスト集合のグループ分けを行う．その後バンド幅の干渉の有無を用いてリンクを共有するグループペアを求め，より上位のスイッチを推定する．先に RTT によってグループわけをすることでバンド幅の測定を減らし，ネットワークへの負荷を減らしながら全体のトポロジを推定できる．しかし，この手法でも RTT を全体全で測定しており，測定に時間がかかってしまう．我々の手法は限られたホストペアの RTT の測定だけを用いて高速に推定する．バンド幅を用いた推定精度向上という方向性は十分が考えられるが，本論文では RTT 測定だけでも高い精度を得られることを示す．

Lowkamp らはネットワークの情報の表現方法について提案した [16]．この研究の焦点は表現方法のより一般的なフレームワークの提供であり，トポロジーの推定については述べられていない．我々の目的とするトポロジー推定はこのような研究を補間するものといえる．

第3章　RTTの測定にもとづいたトポロジー推定

3.1　トポロジー推定システムの要求

我々が想定する大規模並列計算環境での推定は，以下の要求を満たすことが望まれる．

- 手動設定が不要
  現在普及しつつあるマルチクラスタ環境では，SNMP などのをそれぞれのクラスタで個別に設定することは大きな手間となり，また管理者権限を持たない一般ユーザには不可能であることもあるため，ネットワーク管理情報を用いることはできない．一方でこのような環境でよく用いられる情報として，ホスト名や IP アドレスなどによるグループ分けがある．しかし，今後構築されるであろう，より広域に分散したホスト群上に仮想 LAN を構築したような環境では，ホスト名や IP アドレスがネットワークの局所性とは無関係になることが考えられる．そこで，SNMP などの管理者が行う設定はもちろん，ホスト名などを含めたいかなる設定も必要としない環境適応型の手法が望まれる．さらに，システムが用いるパラメータは事前に与えられることなく，測定結果から導く必要がある．

- ネットワーク的にヘテロな環境で動作
  LAN のような局所的な環境から，WAN のような広域に分散した環境まで適用するために，ネットワーク的にヘテロな環境で動作することが望まれる．

- 高速な推定
  多数のユーザが使用するなどの理由で使用可能なリソース頻繁に増減する動的な環境では，処理の開始時までユーザが用いる計算リソースを決定できない．例えば科学技術計算を行う場合は，計算開始時に CPU 負荷が低いホストを選択する．そこで，ホスト群を選択してから推定を行い，その後計算を開始するという使い方が想定される．したがって，推定自体にかかる時間が非常に短いことが望まれる．

- 低負荷
  多数のユーザがリソースを共有している環境では，推定システムが他のユーザの処理に影響を与えることは望ましくない．また他のユーザのプロセスの影響によって性能が低下することも

望ましくない．したがって，推定システムはネットワーク的な負荷や CPU 負荷が低いことが望まれる．例えばバンド幅の測定やパケットロス率の測定は長時間の測定が必要になり，さらにネットワーク負荷が高く他のユーザに迷惑をかけてしまう．

これらの要求を満たすために，我々の手法はエンドホスト間で小さいメッセージの送受信し，その往復時間の測定結果だけを用いて推定する．そして最尤推定のような確率的手法を用いず，よりシンプルで高速なアルゴリズムでツリートポロジーを決定する．

この節では，実際のトポロジーがツリーであり，かつトポロジー上のリンクで生じる遅延はそのリンクを通るどのホスト間で測定しても同じになる，という条件 (∗1) を満たす理想的な状態での推定アルゴリズムについて述べる．測定の変動がなければ，別々のホスト間の通信であっても同じケーブルやスイッチ，ルータでは同じ遅延が生じるという仮定は十分に通用する．測定するホストペアの選択方法や測定の変動の考慮，全ホストでの動作方法といった，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 節で述べる．

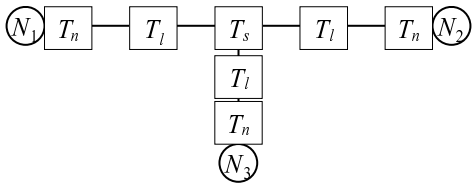
3.2 ツリーモデルについての議論

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単純化（抽象化）と一般化の妥協点として，ツリーがよい表現であると考えている．単純化という面では，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する．そして，ツリーはそのまま階層的なクラスタリングに用いることができる．ホストを階層的にクラスタリングするということは多くのアプリケーションにとって有用である．またトポロジーをツリーに制限することで，推定を単純で高速にすることができる．一般化という面ではまず LAN 内のネットワーク，特にイーサネットではトポロジーは通常ツリー構造である．WAN ではツリーという制限を行うことで，通信路が冗長化された環境では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は多く存在する．

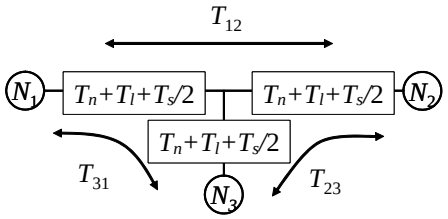
3.3 ネットワークトポロジーモデル

我々の手法では，ネットワークをツリー構造にモデル化する．リーフ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（ルータやハブなどを含む）を表す．また，リンクには遅延を表す重みをおく．

現実にはパケットの遅延は図 3.1(a) のように，ホストでの遅延（T_n），ケーブルでの遅延（T_l），スイッチでの遅延（T_s）に分けられる．しかし，ホスト間の RTT の測定によってこれらの個々の遅延を区別することは不可能である．我々は，図 3.1(b) のようにこれらの遅延をまとめることによって，



(a) ホスト，ケーブル，スイッチによる現実のの遅延モデル



(b) リンクだけに遅延を持たせた本手法におけるモデル

図 3.1: ネットワークモデル

ネットワークをリンクにのみ遅延が発生するモデルとして扱う．このような形にすることで, (∗1) を満たすトポロジーを一意に求めることができる．

3.4 トポロジー推定アルゴリズム

(∗1) が成立するときはホスト間の RTT とツリーが一対一で対応する．したがって，本節で述べるアルゴリズムは推定というよりは，ホストペアの遅延から一意のツリーを構築するアルゴリズムといえる．本節では，全ホストペアの RTT を必要とせずに前節で述べたモデルのツリーを構築する手法について述べる．

3.4.1 3 ホストのトポロジー

我々の手法は，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加えていくことで全体のトポロジーを推定する．3 ホストのトポロジーの形は図 3.2 のように一意に決まる．それぞれの枝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．

$$\begin{aligned} x &= (\mathrm{AC} + \mathrm{AB} - \mathrm{BC})/4, \\ y &= (\mathrm{AB} + \mathrm{BC} - \mathrm{AC})/4, \\ z &= (\mathrm{BC} + \mathrm{AC} - \mathrm{AB})/4, \end{aligned} \tag{3.1}$$

AB，BC，CA はそれぞれのホスト間の RTT の測定値とする．[1]

[1]RTT はホスト間の遅延の 2 倍の値であるので，この方程式の係数は 1/2 ではなく，1/4 になる．

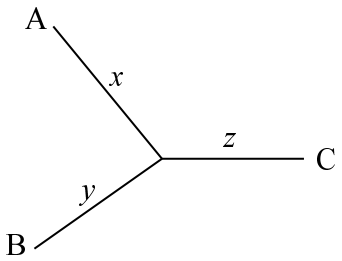


図 3.2: 3 ホストのトポロジー

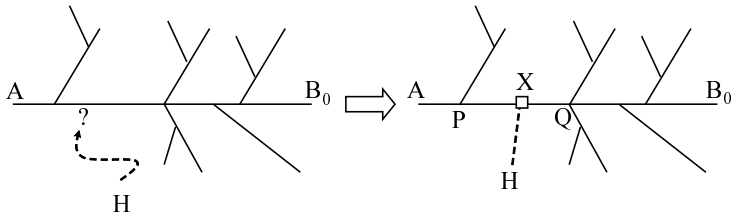
3.4.2 4 ホスト以上のトポロジー

次に 4 ホスト以上のトポロジーを推定する場合について，既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 (3.1) を解く．これにより，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とができる．この分岐点がとりうる位置は以下の 2 つに分かれる．

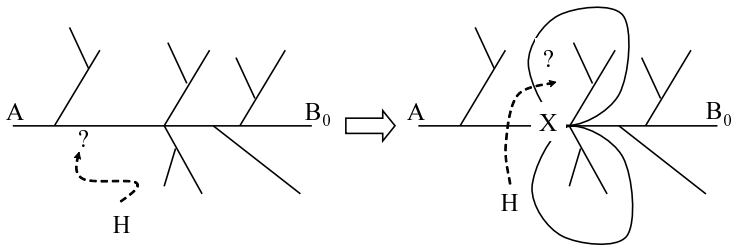
1. 図 3.3(a) のように推定されたツリー上では分岐点ではない場合，新しい分岐 X を加える．ここでの X はスイッチを表す．そして X に新しいリンクを加えて，その先に H を加える．X から H までの遅延は式 (3.1) によって求められたリンクの遅延の値である．

2. 図 3.3(b) のように推定されたツリー上で既に分岐（X とする）である場合，H が X をルートとするサブツリーのうち，A や B_0 を含まないもののどこ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 と交代し，A や B_0 を含むサブツリーのホストを測定対象から除く．そして，A，B_1，H の 3 ホストについて，同様の手順を行う．もちろん B_0 ではなく A と交代してもよい．この選択については 4 節で述べる．これを繰り返すことによって，分岐位置がツリー上ですでに分岐だったときに，測定対称に残る B_0 が存在しなくなることがある．このようになったときは，その分岐に新しくリンクを加え，その先に H を加える．X から H までの遅延は式 (3.1) によって求められたリンクの遅延の値である．

AH と BH の測定によって A，B，H の分岐位置 X を決めることで，H は X をルートとするサブツリーのうち A と B を含むものには加えられないことがわかり，この 2 つのサブツリー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を決定することができる．この手順（`add`）とトポロジー推定全体の手順（`infer`）を図 3.4 に示す．このように，我々の手法は測定結果を用いてホストを追加するときに最尤推定などを用いずに直接位置を決定するため，CPU 負荷も低い．



(a) A，B_0，H の分岐位置がツリー上で分岐ではない



(b) A，B_0，H の分岐位置がツリー上で分岐している

図 3.3: 2 ホストとの測定で得られる情報

3.4.3 RTT とツリーが一対一に対応することの証明

この項では（∗1) が成立するときは元のネットワークとホスト間の RTT と推定されるツリーが一対一で対応することを示す．まず,（$ast1$）が成立するときはホスト間の通信路が一定で，各リンクやスイッチでの遅延も一定であるので，測定される RTT も一定である．次に，推定されるツリーが一意に決まることを示す．3.4.1 項で述べたように，3 ホストのトポロジーは RTT から一意に決まる．したがって，もとのネットワークと一致する．

4 ホスト以上については，追加したホスト群の位置が元のネットワークに一致する時に，新しくホストを追加するとそのホストも元のネットワークの位置と一致することを示す．まず前項の 1 のようにリンクに新しい分岐 X を加える場合を考える．A と B_0 の分岐位置 X は 3 ホストのときと同様に一意に決まる．さらに，図 3.3(a) のように，X を加える分岐がスイッチ P と Q にはさまれている（P が A の側，Q が B_0 の側）とすると，以下の式が成立する．

$$\begin{aligned} AH &= (\mathrm{AP} + \mathrm{PX} + \mathrm{XH}), \\ BH &= (\mathrm{BQ} + \mathrm{BX} + \mathrm{XH}), \\ AB &= (\mathrm{AP} + \mathrm{PX} + \mathrm{XQ} + \mathrm{BQ}), \end{aligned} \tag{3.2}$$

AH，BH，AB は測定によって一意に決まる．また AP，BQ はそのパス上の遅延の和によって一意にもとまる．したがって，PX，BX，XH は一意に決まる．さらに，各ホストから P や Q までの遅延は元のネットワークと一致していることから，各ホストから X までの遅延も元のツリーに対応す

る RTT と一致する．

次に，前項の 2 のようにツリー上にすでにある分岐にホストを加える場合を考える．A と B_0 の分岐位置 X は 3 ホストのときと同様に一意に決まる．さらに，XH は式 (3.1) から一意にもとまる．各ホストから X までの遅延も元のツリーと一致していることから，各ホストから X までの遅延も元のツリーに対応する RTT と一致する．よって, $(*1)$ が成立するときは元のネットワークと推定されるツリーが一対一で対応することが帰納的に示された．

3.5 測定回数の上限

ホストの数を N，推定されたツリー上の全ホスト間のホップ数の最大値（ツリーの直径）を d，一つのスイッチのポート数を p とする．この時，手順 `add` の繰り返しの回数が pd 以下になり $(*2)$，推定全体の手順 `infer` での測定回数が $(pd+1)(N-2)$ 以下になることを示す．p が定数であるとすると，測定回数は $O(Nd)$ となる．

$(*2)$ を示す．H を追加するときのことを考える．A と B を用いた `add` により A または B を含む X をルートとするサブツリーを探索空間から除くことができる．ただし，X 自体は除かれない．次の `add` のための新しいホストを A も B 含まないサブツリーから選択することで，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．また，一回の `add` で少なくとも一つのサブツリーを除くことができるので，X にとどまる回数は最大で p 回である．したがって，`add` の繰り返し回数は pd 以下になる．d は通常 $(O(\log N))$ より小さいため，スケーラビリティにおいてもよい性質を持つ手法であるといえる．

```
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 (3.1);
    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 3.4: トポロジー推定アルゴリズムの基本的な手順

第4章　実環境における測定と実装の詳細

3 章で述べた理想的な環境での推定手法では，手順 `infer` における追加するホストの順序 H_i と手順 `add` における A や B の選択は任意であった．しかし，実際に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ってしまう．本章では実環境で高精度かつ高速に推定するための手法について述べる．

4.1　一対のホスト間の RTT の測定

一対のホスト間の RTT は，送信ホストが数バイトの TCP パケットを送信し，受信ホストはそれを受信したらすぐに返信することで測定される．RTT は送信ホストの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．この上限は今は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つ得る．そして，その中の最小値を RTT の測定値として用いる．このように複数回の ping-pong を行うことで，突発的に生じる大きい遅延に対応する．この手順の疑似コードを図 4.1 に示す．

3 つのパラメータを示したが，測定結果はこれらのパラメータの選択によって大きく変動するものではない．図 4.2 は一つのクラスタ内でそれぞれ２，４，６ホップ離れたホスト（2,4,6 hops）間，及び同じ建物にある別のクラスタのホスト（Cluster 3）との間で測定した RTT の分布を示す．クラスタ内のスイッチは DELL PowerConnect 5224 を用いており，２ホップとは同じスイッチにつながるホスト同士の測定を意味する．クラスタ間には TCP レイヤのルータが３つ存在する．クラスタ内では２ホップ増えると RTT が 10μ 秒ほど増加したが，測定結果の変動は 10μ 秒より小さい．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突発的な大きい遅延の影響を受けることなく，ホスト H と近いホストとの間の RTT とより離れたホストとの間の RTT を区別することができる．

ただし，一部の NIC を用いたホストは RTT の測定結果の変動が非常に大きい．我々が知るものとしては，e1000 で InterruptThrottleRate オプションがオンの場合にそのようなことが起こる．これは，パケットを受信してすぐに OS に通知するのではなく，受信したものをバッファにためて定期的に通知を行うという機能である．この機能は小さなパケットを大量に受信したときに，まとめて通知することで CPU 負荷を低下させることが目的であり，デフォルトでオンになっていた．この

```
measure() {
  /* the sender procedure */
  min = ∞;
  max = −∞;
  for (i = 0; i < 3; i++) {
    best = ∞;
    for (j = 0; j < 30; j++) {
      rtt = single_round_trip();
      if (rtt < best) best = rtt;
      if (best was not updated in the last ten iterations) break;
    }
    if (best > max) max = best;
    if (best < min) min = best;
  }
  return both min and max as the result of the measurement;
}
```

図 4.1: 一つのホストペアの測定手順

オプションが使われている場合と使われていない場合で RTT の測定結果は図 4.3 のようになった．InterruptThrottleRate がオフのときは測定の変動は 5μ 秒程度であるのに対し，オンのときは 550μ 秒と，オフのときの RTT の平均値の 600%の変動が見られた．このようなホストに対しては，RTT の測定を用いた推定を行うことは困難である．

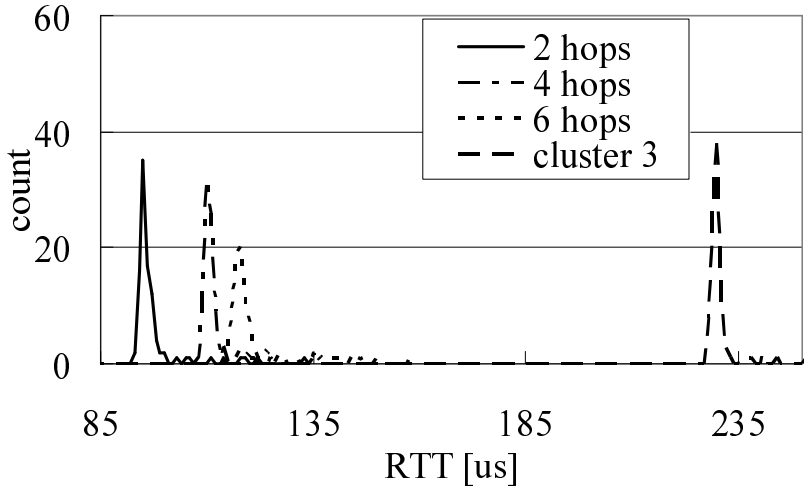
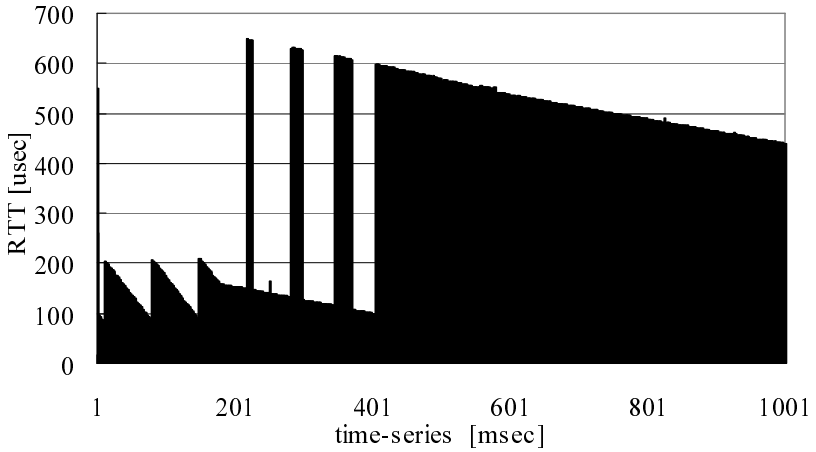
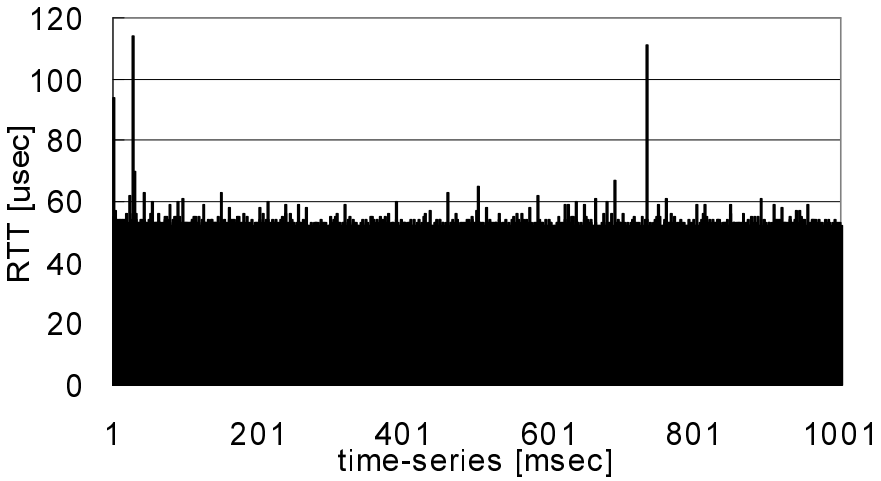


図 4.2: Cluster 4 （NIC は bcm5700）における RTT の分散



(a) InterruptThrottleRate が on のとき



(b) InterruptThrottleRate が off のとき

図 4.3: e1000 を用いたときの RTT の測定結果

4.2 分岐位置の一致範囲

時間の測定に gettimeofday などのシステムコールを用いた場合，その返り値はマイクロ秒単位で量子化されたものである．そこで，手順 add の図 3.3(b) において分岐位置が一致する，ということをマイクロ秒単位の一致とみなすことができる．しかし，現実のネットワークでは測定値に変動があり，そのように一致することはほとんどない．そこで，分岐位置の差がある範囲内に収まった場合に分岐位置が一致したとみなす．この範囲が小さすぎる場合は，同じスイッチにつながるノードが小さな遅延のリンクをもつ複数スイッチに分かれるように推定される．逆に大きすぎる場合は隣接するスイッチがひとつにまとまってしまい，別々のスイッチにつながるはずのノードがひとつのスイッチにつながるように推定される（図 4.4）．推定対象のホストやそのネットワークにあるスイッチによって適切な範囲が異なるため，この範囲も適応的に決める必要がある．

そこで，4.1 節で述べた 3 つの最小値のうちの最大値と最小値の差を分岐位置の範囲とする．推定したツリー上の分岐がその範囲内にあれば，その中で式 (3.1) によって得られる分岐位置に最も近い分岐を H に隣接する分岐として扱う．これは複数回の測定によって得られた測定結果の変動以上の遅延に意味を持たせ，それより小さい遅延は測定におけるノイズであり意味を持たないという考えにもとづく．

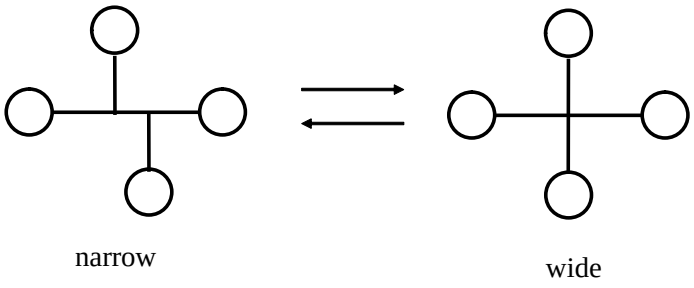
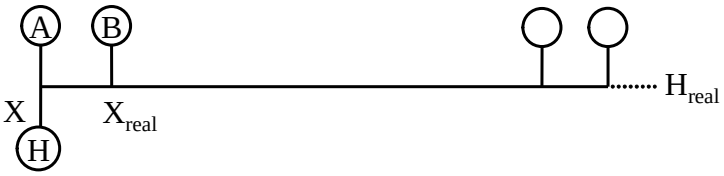


図 4.4: 分岐位置の一致範囲による推定結果の変動

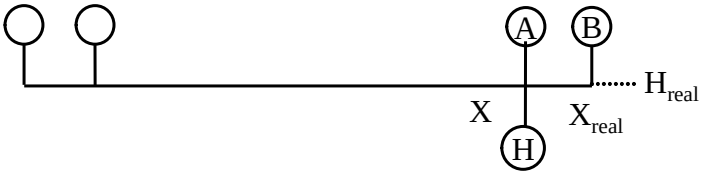
4.3 手順 add で測定するホストの選択

手順 add でホスト H を追加するために，ホスト H から他のホストへの測定を行う．そこでこのアルゴリズムの実装方針として，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法を採用する．ホストを $H_0, H_1, \ldots$ と並べて，ホスト H_i は $H_0, \ldots, H_{i-1}$ を含むツリーの到着を待つ．到着したらツリーに H_i を加えて 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減らすためには，適切なホスト間の測定結果を用いることが重要になる．例えば図 4.2 のような環境では 90μ 秒と 100μ 秒の差を区別できなければならない．近いホスト間の測定結果の変動は遠いホスト間の測定結果の変動より小さいと期待される．また，図 4.5 のように，近いホストを用いて分岐位置を間違えても実際の位置から大きく離れることにはあまりならないが，遠いホストを用いて分岐位置を間違えると実際の位置から大きく離れてしまう恐れがある．したがって，手順 add で A と B を選ぶときに H に近いホストを選ぶことで測定精度が向上する．これにはもう一つ，手順 add での探索空間を大きく削減するという効果もある．初めに近いホストを選択すればツリーへの追加を素早く行うことができる．



(a) 遠いホストを選択したとき



(b) 近いホストを選択したとき

図 4.5: 測定の変動による誤りの大きさ

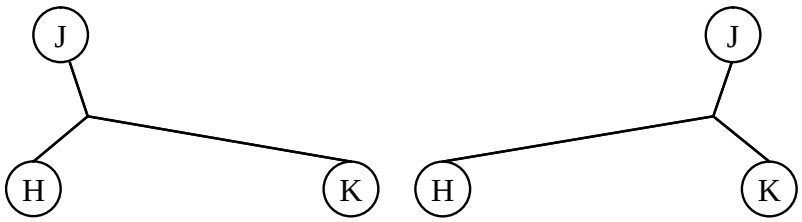
これらを実現するために，２つの方法を用いる．

- add のイテレーションの最初の A の選択を選択するときに H にできるだけ近いホストを選択する．そのための測定はツリーに一つずつホストを追加するという手順と並行して行い，追加に使うツリーに先行してもう一つのツリーを用いる．
- 手順 add のイテレーションの最後（“A = either A or B”）に X に近いホストを用いる．これは前者が成立する場合，分岐 X に近いホストが H に近いことが多いためである．

前者についてより具体的に述べると，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 が十分に大きくなると T_i は H_{i+10} が後に受け取る H_{i+9} のよい近似となる．基本的には T_i に含まれるホストをランダムに選択し測定を行い，最も近いホストを求める．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，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一つずつしか測定しない．この手順を 4.7 に示す．

H_i をツリーに追加するときの，これらを組み合わせた手順を 4.7 に示す．



(a) $JK > 3x$ (b) $JK < x/3$

図 4.6: ホスト K を測定候補からはずす基準

4.4 システムの起動方法

我々のシステムは各ホストでプロセスを起動し，それぞれのホストの IP アドレスとポート番号を共有する必要がある．このシステムの起動には GXP[19] を用いた．GXP はグリッド用のシェルであり，プロセスを起動するホストの選択，複数ホストでのプロセス起動，実行時の ssh 通信網を用いた簡易メッセージパッシング機能を有す．

GXP を用いて各ホストでのプロセス起動を行い，GXP の通信機能を用いて IP アドレスとポート番号の共有を行う．その後は共有したポート番号を用いた通信によって測定およびツリーの送信を行う．実行時に IP アドレスとポート番号を共有し，さらに推定に用いるパラメータを適応的にしたため，我々のシステムにおける起動時の設定は対象ホストの選択のみになる．対象ホストの選択は並列処理を行う際にも必要であることが多いため，事実上我々のシステムに必要な設定はない．

```
infer_local() {
  /* the local procedure for topology inference */
  wait for a tree T'(= T_{i-10}) from H_{i-10};
  A = find_close_host(T');
  wait for a tree T(= T_{i-1}) from H_{i-1};
  add(T, H_i), with choosing A as the first host;
  send T to H_{i+1} and H_{i+10};
}

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 4.7: 測定手法を含めた一つのホストにおける推定手順

第5章　評価

5.1　実験環境

我々の推定手法を C++言語で実装し，図 5.1 のような 4 クラスタ 256 ホストの環境で実験を行った．Cluster 4 については，左の 2 つのスイッチにつながるホストのうちの 64 ホストを用いた．各クラスタ内はレイヤ 2 スイッチで構成されており，図 5.1 で示したスイッチ以外は存在しない．クラスタ間はレイヤ 3 ルータ構成されており，図 5.1 では分岐を構成するルータだけ示されている．また図中の時間はクラスタ間およびクラスタ内で測定される遅延を示す．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．ネットワークインタフェースは全て Gigabit Ethernet である．Cluster 3 と 4 は同じ建物にあり，Cluster 1 は 25km ほど，Cluster 2 は 31km ほど Cluster3 や 4 と離れた位置にある．このように，クラスタ間の遅延はそのペアによって異なる値が測定される．

5.2　推定精度

5.2.1　推定結果の図を用いた定性的な評価

精度についてのイメージとして，まず我々の手法を用いて推定されたツリーを図 5.2 に示す．同じ色のホスト群は同じスイッチにつながるということを意味する．この図では Cluster 1 は左上に，Cluster 2 は左下に，Cluster 3 は右上に，そして Cluster 4 は右下に，それぞれ分かれて推定された．また同じ色のホストがつながっていることから，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たことが分かる，さらにより上位のスイッチについても部分的には推定できた．例えば Cluster 2 のホスト群は左下に示されており，4 つのホスト群が十字に分かれた．また，Cluster 3 のホスト群は右上に示されており，5 つのスイッチにそれぞれホストがつながり，一列に並んだ．

ただし，グループ間の接続関係は必ずしも全ては推定できていない．まずクラスタ間については，我々の手法では多くの場合 Cluster 1，2 と Cluster 3，4 をつなぐ遅延 $5\mu s$ のリンクを推定することができなかった．これは，Cluster 1 や Cluster 2 へのリンクの遅延が三桁大きいため，それらの遅延の変動によって $5\mu s$ が隠されてしまったためである．このように，遅延の大きいリンクで囲まれた遅延の小さいリンクを推定することが困難であるという課題が残る．また，クラスタ内でもスイッチ間のリンクは正確には推定できなかった．

	OS	CPU	NIC driver
Cluster 1	Linux Kernel 2.6	Dual Xeon 2.4GHz	e1000
Cluster 2	Linux Kernel 2.6	Pentium M 1.8GHz	sk98lin
Cluster 3	Linux Kernel 2.6	Pentium M 1.8GHz	sk98lin
Cluster 4	Linux Kernel 2.4/2.6	Dual Xeon 2.4/2.8GHz	bcm5700

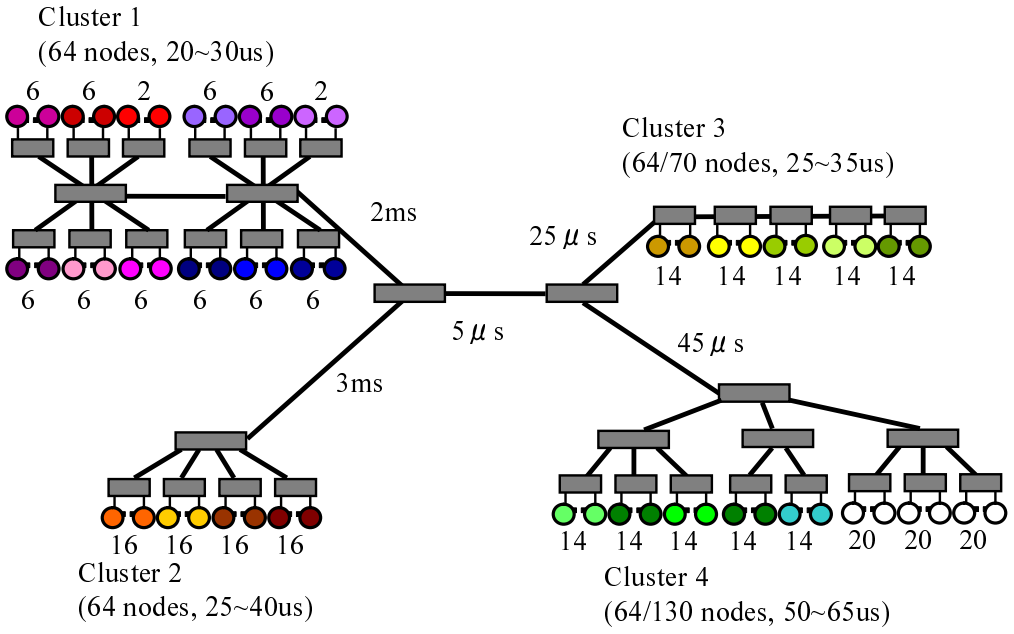


図 5.1: 実験環境

5.2.2 分岐位置の区間

Cluster 1 と 4 では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たが，一つのスイッチにつながるのではなくいくつかのスイッチに分離してしまった．これは 4.2 節で述べた，分岐位置の一致範囲が狭かったためである．図 5.3 はこのホスト群の分離を手動で訂正したものである．具体的には，推定されたトポロジー上の隣接する 2 つのスイッチにつながるすべてのホストが，実際のトポロジー上で同じスイッチにつながるときだけ 2 つのスイッチをマージするという処理を行った．図 5.3 では Cluster 1 や 4 で同じスイッチにつながるホスト群がより正確にグループ分けされることが分かる．

Cluster 1 や 4 ではスイッチ間のリンクの遅延が約 $7\mu s$ であるのに対し，分岐位置の区間は $2-4\mu s$ 程度であった．一方 Cluster 2 や 3 ではスイッチ間リンクの遅延が $4\mu s$ であるのに対し，分岐位置の区間は $1-2\mu s$ 程度であった．この区間の大きさによる推定結果の変動を示すために，分岐位置の区

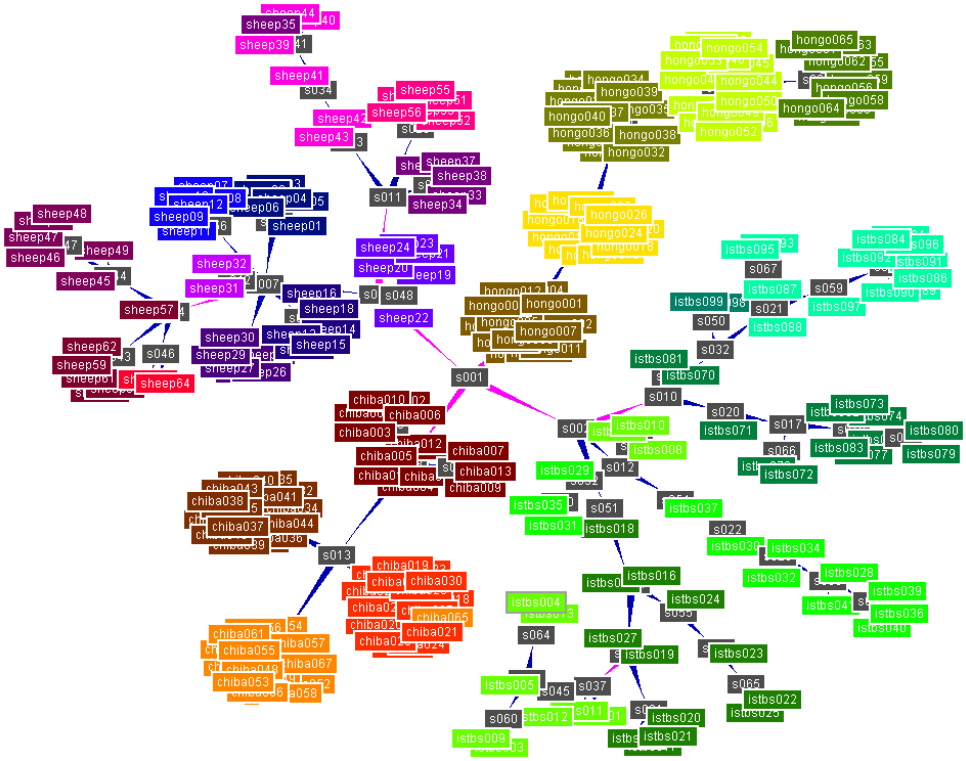


図 5.2: 4 クラスタ 256 ホストで推定したトポロジー

間を静的に決定して推定を行った．図 5.4 は Cluster 3 と Cluster 4 での推定結果である．Cluster 3 は区間 4μs のときにホスト群の分離がなくなり，区間 5μs 以上では異なるスイッチにつながるホスト群が同じスイッチにマージされてしまった．一方 Cluster 4 では区間 7μs のときに最も高い精度でグループ分けでき，8μs 以上では別々のスイッチにつながるホスト群が同じスイッチにマージされてしまった．このようにホストやスイッチによって適切な値が異なるため，この値を静的に決めることはできない．4.2 節で述べた，測定の変動以上の遅延に意味を持たせ，それより小さい遅延を無視するという方法はある程度の合理性があると考えているが，さらに適切にグループ分けを行う指標が求められる．

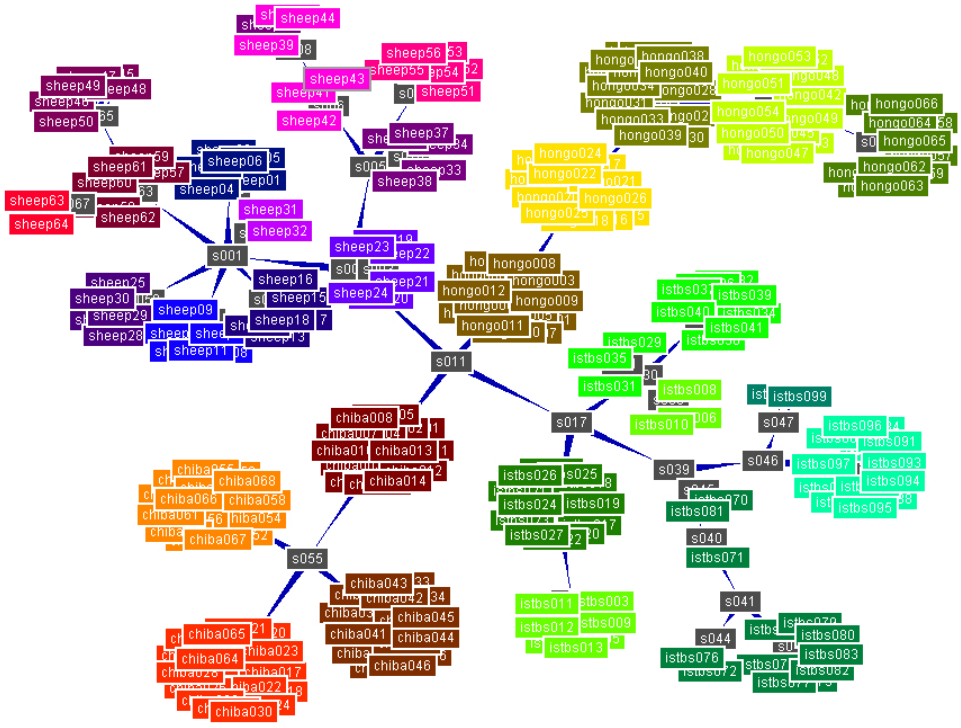
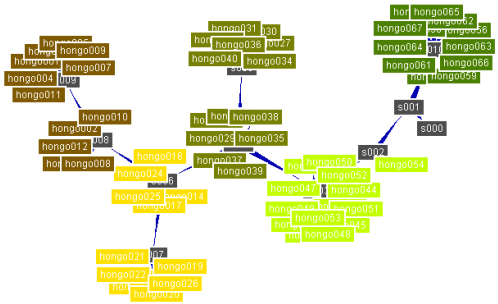
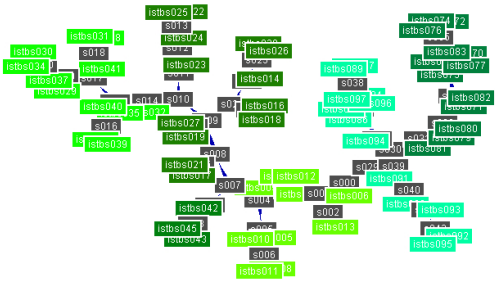


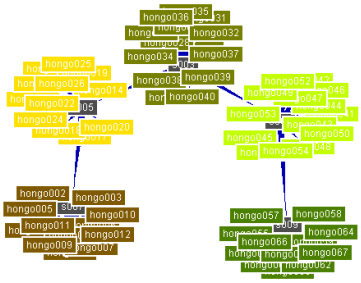
図 5.3: 4 クラスタ 256 ホストでスイッチ単位でマージしたトポロジー



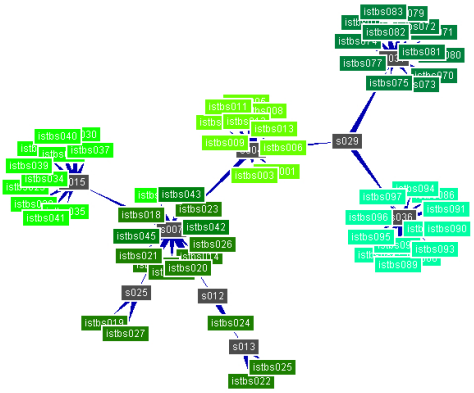
(a) Cluster 3, 区間 1μs



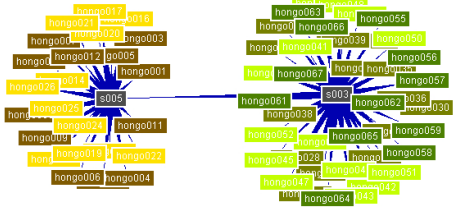
(b) Cluster 4, 区間 1μs



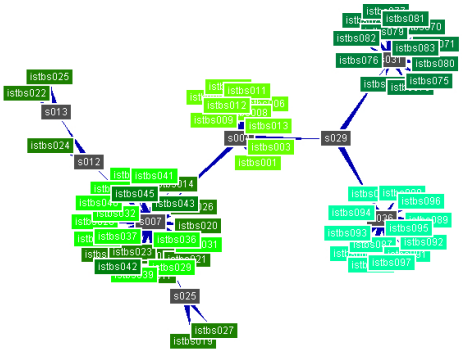
(a) Cluster 3, 区間 4μs



(b) Cluster 4, 区間 7μs



(a) Cluster 3, 区間 5μs

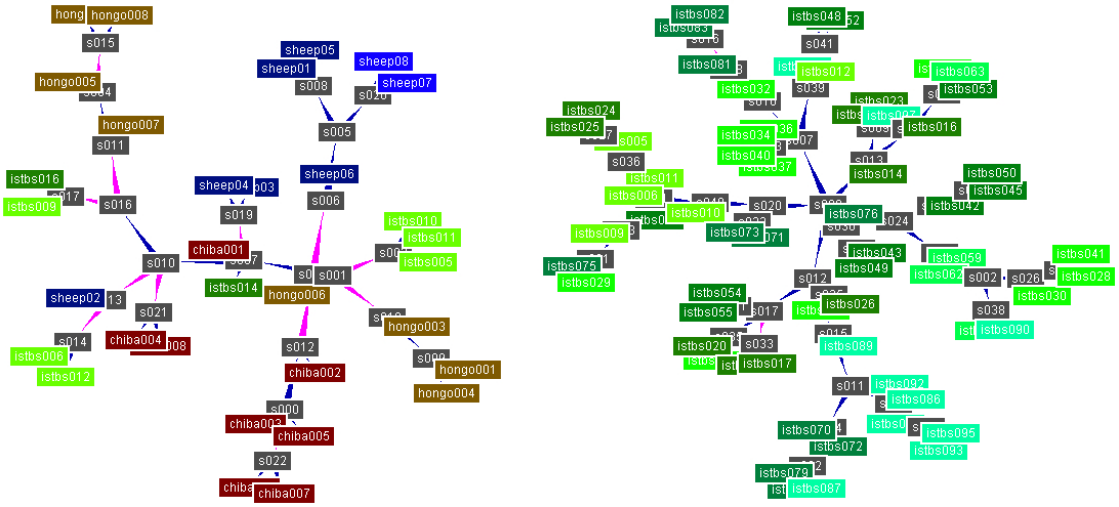


(b) Cluster 4, 区間 8μs

図 5.4: 分岐位置の区間の変動にともなう推定結果の変動

5.2.3 近いホストの選択の効果

4.3 節で述べた，近いホストの選択を行わない場合の推定結果を図 5.5 に示す．まず図 5.5(a) では，クラスタのグループ分けがうまくいかなかった．これは遠いホスト同士の変動の大きい測定結果から追加する位置を決定してしまい，さらに誤った位置に推定されたホストとの測定を用いて新たなホストの推定をすることによって誤りが広がり，このようにクラスタ間でも誤ってしまったと考えられる．また，図 5.5(b) から，単一のクラスタ内でもホップ数の大きいホストとの測定結果を用いると追加位置を大きく誤ってしまうことが分かる．以上の結果から，できるだけ近くのホストを選択し，ホストの追加の時には変動の小さい測定結果を用いることが非常に重要であることがわかる．



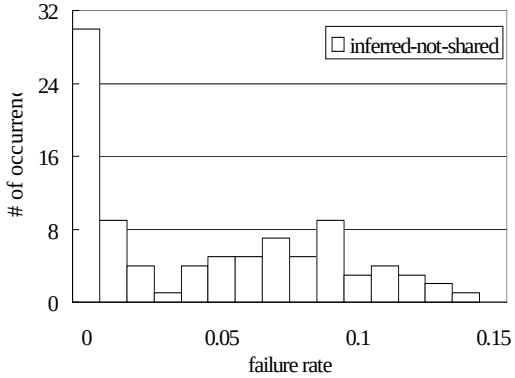
(a) 4 Cluster 32 ホスト　　(b) Cluster 4，64 ホスト

図 5.5: 測定するホストをランダムに選択した場合の推定結果

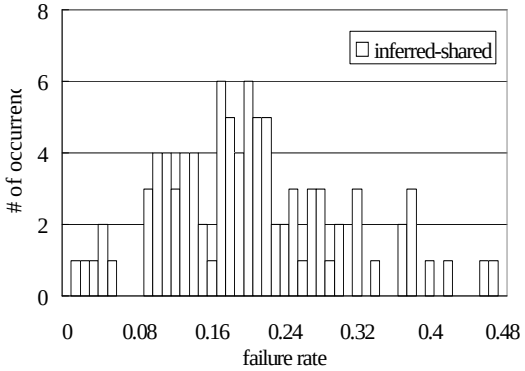
5.2.4 共有リンクの問い合わせを指標とした定量的な評価

推定されたトポロジーの精度を定量的に評価する指標として ”A と B，C と D の通信路が一つのリンクを共有するかどうか ”という問い合わせへの回答の正誤を用いる．ここでは 2 つの通信路が同じリンク一つでも通るときに 2 つの通信路が一つのリンクを共有すると考える．実際の物理トポロジーから得られる回答と推定したトポロジーから得られる回答を比較した．一つのトポロジーにつき 10 万回ランダムに 2 つのホストペアを選択して問い合わせを行い，誤り率を求めた．我々のトポロジー推定手法は決定的ではないため，図 5.6 には Cluster 4 および全クラスタで 100 回推定を行った結果についての誤り率の分布を示す．左の図は推定結果から共有リンクがあるという回答が

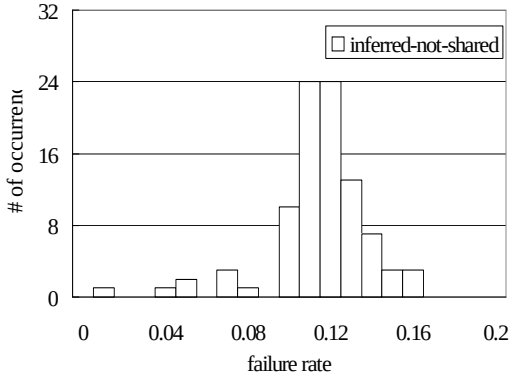
得られたが，実際は共有リンクがない場合の誤り率（false positive）である．右の図はその逆（false negative）である．単一のクラスタでの false positive 以外では誤り率は 0.2 以下となった．４クラスタでは 5.2.1 項で述べたような，Cluster 1，2 と Cluster 3，4 の間の 5μs のリンクを推定できなかったときに，クラスタ間通信の false negative が高かった．単一のクラスタでは 5.2.1 項で述べたような一つのスイッチにつながるホスト群の分離が起きたときに，false positive が高かったが，それ以外の誤りは少なく，false negative は非常に低かった．



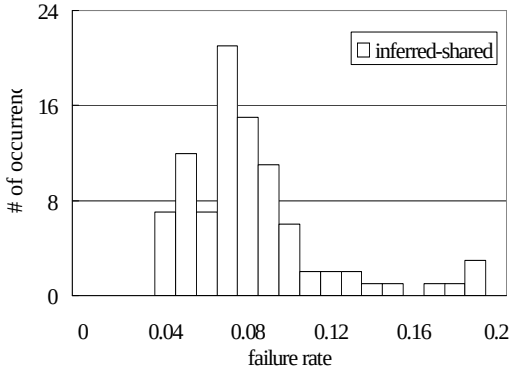
(a) Cluster 4 での false positive



(b) Cluster 4 での false negative



(c) ４クラスタでの false positive



(d) ４クラスタでの false negative

図 5.6: リンク共有の問い合わせに関する誤答率の分布

5.3 推定時間

我々の手法を用いてクラスタ数およびホスト数を変更してトポロジーを推定し，かかった時間を調べた．複数クラスタのときは各クラスタで同じ数のホストを用いた．また 4.3 項で述べた離れたホストとの測定を制限しない場合（non-reduction，4 クラスタ 256 ホスト）との比較を行う．non-reduction ではホスト H_i は T_{i-10} を待たずに，推定開始時から $H_j(j = 1, 2, \ldots, i)$ との測定を行なう．これらの実験結果を図 5.7 に示す．我々の手法の推定時間は 1 クラスタ 64 ホストで 5 秒程度，4 クラスタ 256 ホストで 15 秒程度であった．また同じクラスタ数ではホスト数が多くなってもそれほど推定時間が増加せず，高いスケーラビリティを有することが分かった．一方 non-reduction では 160 ホスト以上で我々の推定手法との差が広がり始め，192 ホストで 20%，256 ホストで 50% 増加した．そしてホスト数が増加すると，この差はますます広がることが予想される．このことから，測定するホストの制限，特に離れたホストとの測定を制限したことによって高速に推定できることが分かる．

我々の手法で 4 クラスタ 256 ホストで推定したときの各クラスタペアまたはクラスタ内で測定を行った回数を図 5.8 に示す．Cluster 1 は同じ Cluster 1 のホストとは全体 $64 \times 63/2$ ペアのうち，38%と測定した．しかし他のホストとは全体 $64 \times 64/2$ ペアのうちの 2%程度とごくわずかとしか測定しなかった．離れたホストとの測定は RTT 自体が大きいために時間がかかる．さらに WAN をまたいで多数の接続を張ることも大きなコストとなる．4.3 項の手法を用いることで，このような比較的コストの大きい測定を減らすことができた．

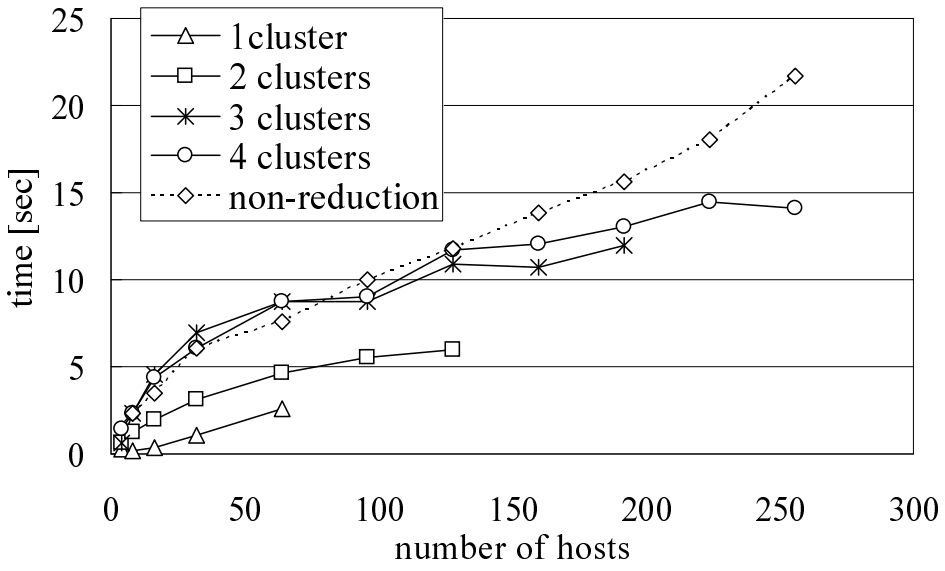


図 5.7: 推定時間

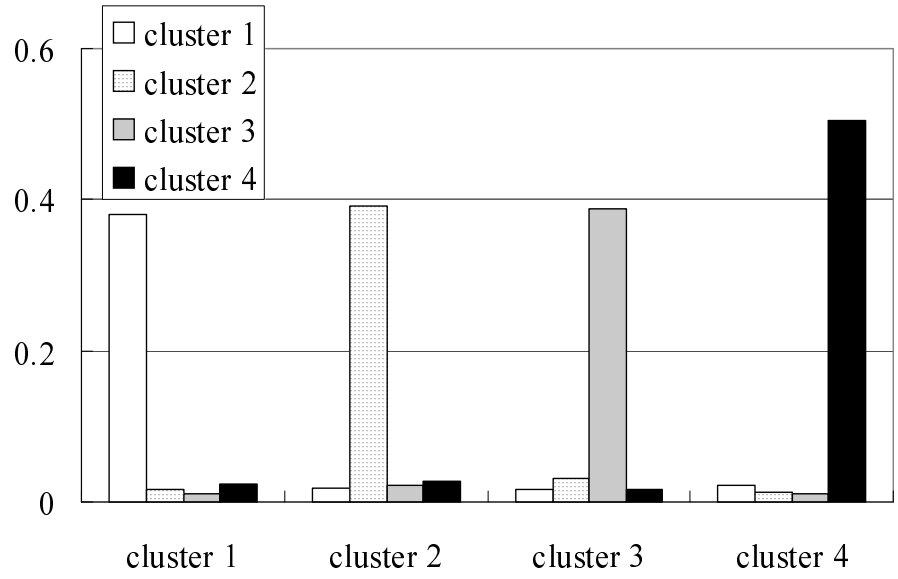


図 5.8: 測定されたホストペアの割合

第6章　トポロジーを用いたアプリケーション

この章では，トポロジー情報を用いてアプリケーションの通信効率を向上させる手法について述べる．まずネットワークの構成そのものが重要になる例として，測定の並列化によるバンド幅ツリーの効率的な構築について述べる．次に複数ホスト間で同時に通信したときのリンク共有情報が重要となる，長いメッセージのブロードキャストの最適化について述べる．そして，より高度なアプリケーションの最適化として，MPI アプリケーションのプロセス割り当て手法について述べる．

6.1　バンド幅ツリーの構築

6.1.1　効率的な測定手法

リンクの重みにバンド幅を与えたツリーをバンド幅ツリーと呼ぶ．バンド幅を並列に測定する場合に，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ー情報を用いない場合は，全ホストペアのバンド幅を得るためには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，バンド幅ツリー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一つ選び，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一つのホストを用いる．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．H_i と H_j との測定によって得られたバンド幅を XC_i 間と XC_j 間のリンクのバンド幅とする．このとき $\{\mathrm{H}_i\}$ 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う．もし p が奇数であれば，先ほどの測定が終わった後で残ったホストと任意のホストとで測定を行う．そのため一ステージで一つのホストは最大 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 6.1 のようになる．stage$1-1$，$1-2$ は同じステージでの一度目の測定と二度目の測定を意味する．スイッチのサブツリーから選択したホストとの測定を用いるという手順を用いる

と，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よ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，さらにツリー上でより上位の，特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ツリーを構築する手法についても研究している．

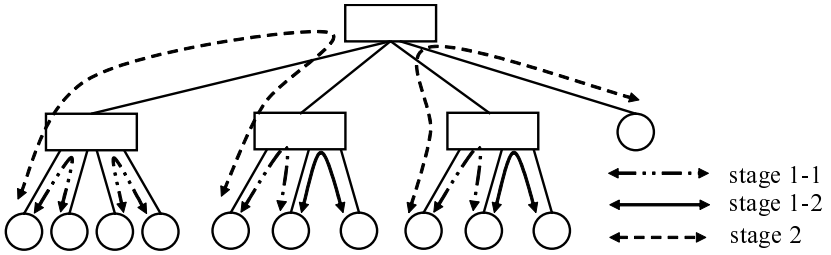


図 6.1: バンド幅ツリー構築のスケジューリング例

6.1.2 測定時間

この手法は深さに比例した時間でバンド幅ツリーを構築できる．ホスト数を N としたときツリーの深さが $(O(\log N))$ であることから，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ツリーの構築にかかった時間を図 6.2 に示す．４クラスタ 256 ホストの環境で 30 秒程度でバンド幅ツリーを構築できた．またホスト数の増加にたいして構築時間はそれほど増えなかった．この実験結果からも，このバンド幅ツリー構築手法が高いスケーラビリティを獲られることが分かる．

6.1.3 バンド幅の問い合わせを指標とした定量的な精度の評価

バンド幅ツリーの精度の定量的な評価として，ホスト間のバンド幅の問い合わせに対する回答と実際に得られるバンド幅を比較した．バンド幅ツリーから得られるバンド幅は，バンド幅ツリー上の通信路のうちの最小のものを用いる．推定したトポロジーから得られるバンド幅と iperf で測定した実際のバンド幅を比較することで得られるバンド幅の正確さを評価する．図 6.3 にバンド幅ツリーから得られたバンド幅と iperf のバンド幅との比の分布を示す．1 クラスタではほぼ正確にバンド幅が得られた．これは，まずクラスタ内のリンクは同じバンド幅であることと，推定結果に多少の誤りがあっても同時に一つのリンクを用いることがなければバンド幅測定に影響を与えないためである．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 5.2.4 項と同様に Cluster 1,2 と Cluster 3,4 の間の 5μs のリンクを推定できなかったことが原因であった．図 5.2(a) では Cluster 1 と 3 の間と 2 と 4 の間で測定が行われ，約 300Mbps という測定結果

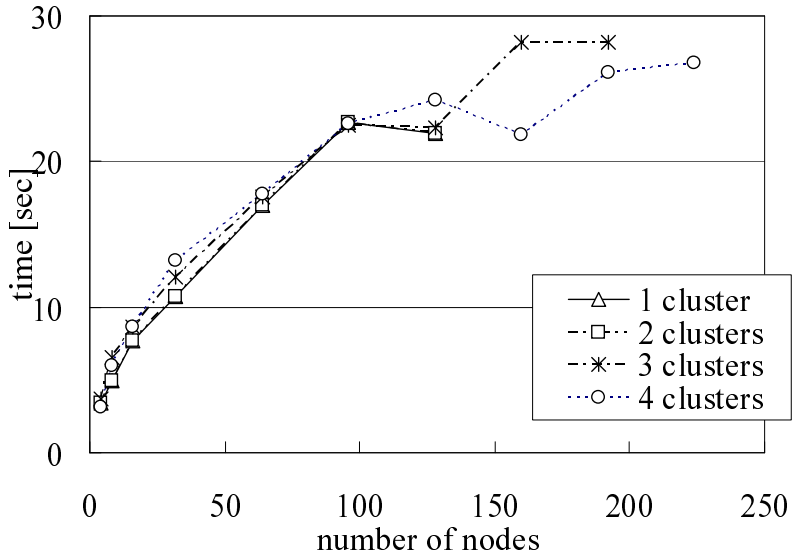
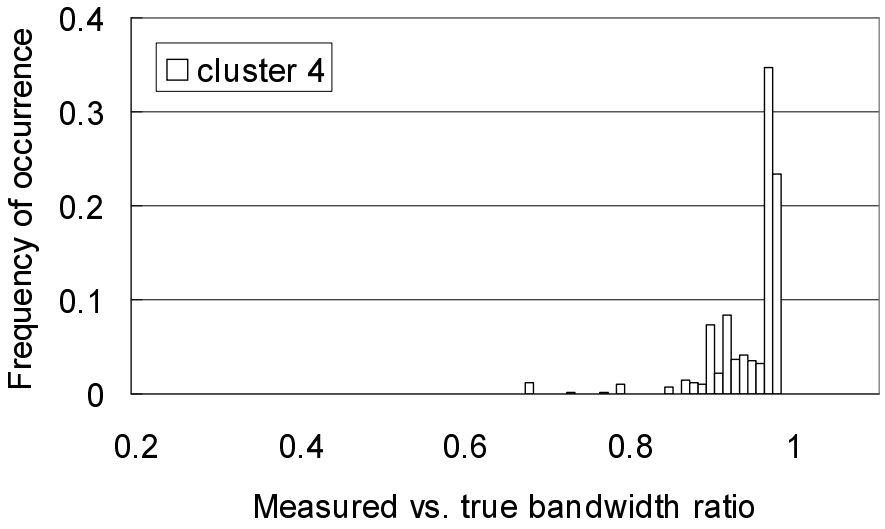


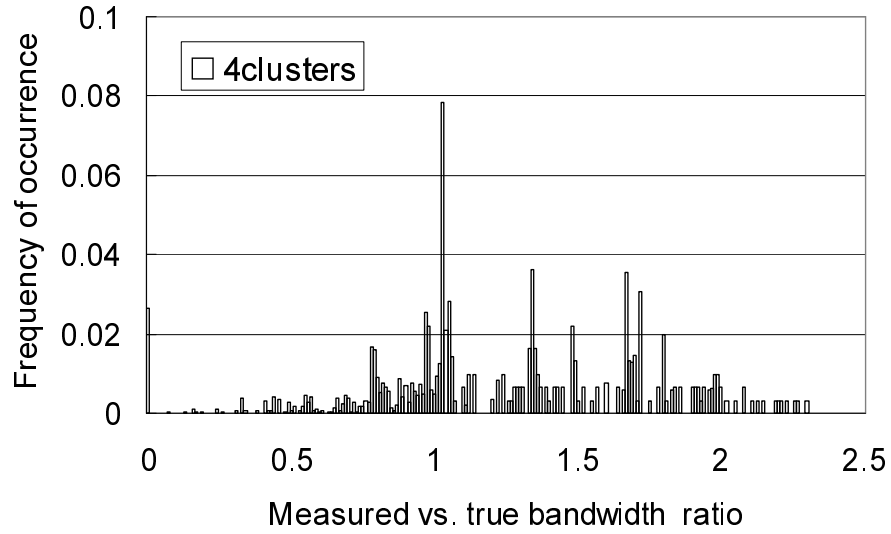
図 6.2: バンド幅ツリーの構築にかかる時間

が得られるため，Cluster 1 と Cluster 2 の間のバンド幅を 300Mbps と回答する．しかし Cluster1-2 間は遅延が $2+3ms$ 程度と大きく，直接の測定ではウィンドウサイズが小さすぎて 100Mbps 程度しか得られない．Cluster 1 と 2 のホストのウィンドウサイズを大きくすることで約 300Mbps のバンド幅を得られた．ホスト間のバンド幅がリンクのバンド幅以外の要素によって制限されている場合は，リンクのバンド幅の測定を組み合わせてホスト間のバンド幅を推定することは難しく，本論文ではこれ以上言及しない．

一般的には全ホスト間のバンド幅を測定することは現実的ではなく，並列に測定できるホストペアを求めて測定する必要がある．本手法はトポロジー情報を用いて $O(logN)$ 時間で近似的なバンド幅ツリーを構築することができた．



(a) 1 クラスタ



(b) 4 クラスタ

図 6.3: バンド幅ツリーの精度

6.2 長いメッセージのブロードキャスト

6.2.1 最適化手法

大きいメッセージのブロードキャストはバンド幅についての最適化が必要である．実際のトポロジーがツリー構造のときは，各リンクをそれぞれ片方向につき一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ー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．実際のトポロジーがツリーでないときは [7] 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を考える必要がある．それでも，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ー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成すことが多い．そのような環境では我々の手法は効果的である．

6.2.2 評価

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．それぞれのホストやスイッチはギガビットイーサネットで接続されており，スイッチ間は 4 本のリンクによるトランクを成す．我々の手法（ours）との比較対象として，実際のトポロジー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 6.4 にこれらのアルゴリズムでの結果を示す．optimal は 624Mbps のバンド幅が得られ，それにたいして我々の手法は 88%の性能を得られた．我々の手法はいくつかのリンクを二度使用してしまうが，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かった．しかし，random はスイッチ間のリンクを多数回使用してしまったため，optimal の 23%の性能しか得られなかった．

6.3 MPI アプリケーション

6.3.1 最適化手法

行列演算のような一部のプロセス間でのみ頻繁に通信するアプリケーションでは，通信するプロセスを近いホストに割り当てることで性能が向上する．MPI モデルではプロセスに Rank と呼ばれる一意の数字を割り当ててプロセスを区別する．一般的に行われる Rank 割り当ては，ホスト名順に割り当てる方法である．これは近い Rank 同士が頻繁に通信するという仮定と，近いホスト名を持つホスト同士がネットワーク的に近い位置にあるという仮定にもとづく．しかし仮想 LAN などの仮想化技術が用いられると，ホスト名とネットワーク的な距離に相関関係を期待できなくなる．そういっ

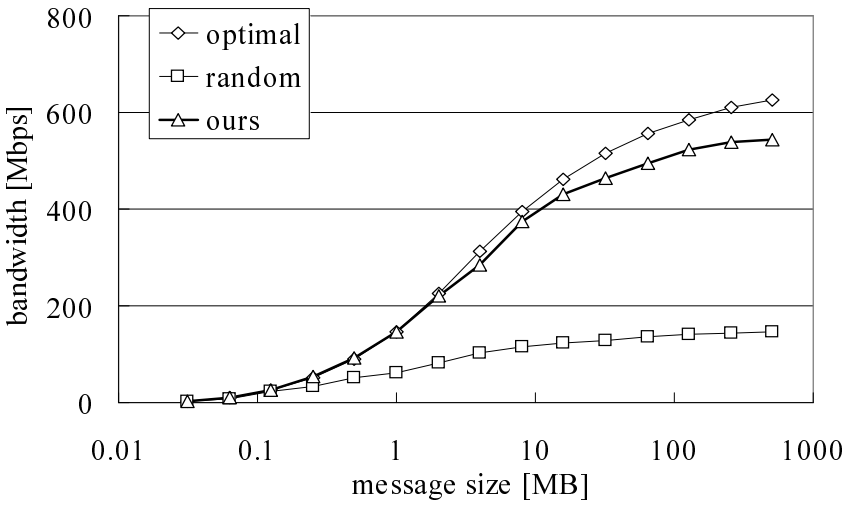


図 6.4: クラスタ 4 内でのブロードキャストで得られたバンド幅

た環境でも，推定したトポロジーにおけるホスト間の距離をもとに Rank 割り当てを行うことにより，物理ネットワーク上距離にもとづく Rank 割り当てを行うことができる．

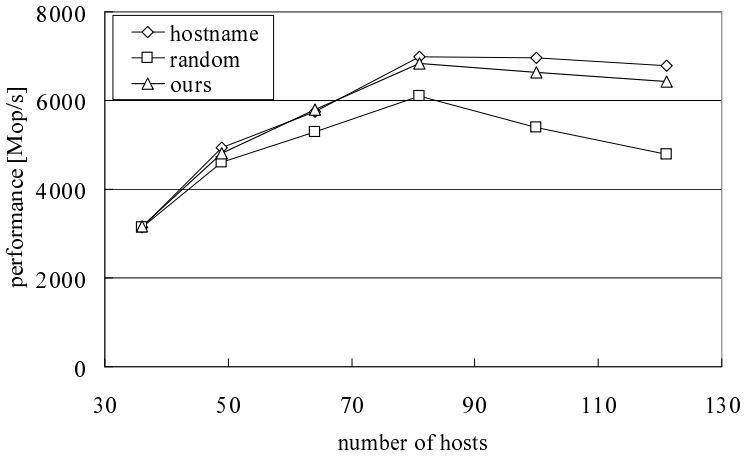
推定したトポロジーを用いて一般的な MPI アプリケーション最適化するためには，アプリケーションの通信パターンの解析を行い，通信パターンに適した Rank 割り当てをする必要があるが，ここでは先ほど述べた，近い Rank 同士が頻繁に通信するアプリケーションを対象とする．推定したトポロジー上で深さ優先順に Rank 割り当てを行うことで近いホストが近い Rank を割り当てられる．

6.3.2 評価

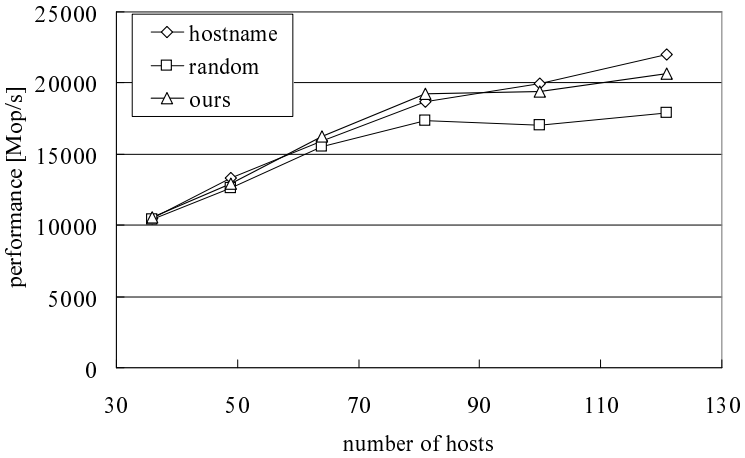
Cluster 4 では近いホストがネットワーク的に近い位置にある．そこで，ホスト名順に Rank 割り当てを行う手法（hostname）とランダム順に Rank 割り当てを行う手法（random）を，推定したトポロジーを用いた手法（ours）と比較する．実験には NAS Parallel Benchmarks の SP，BT，LU を対象とし，実験環境は Cluster 4 の 121 ホストを用いた．結果を図 6.5 に示す．

random は hostname に対して，台数効果が得られる 81 ホストの SP で 13%，BT で 8% 性能が低下した．これに対して ours は 81 ホストの SP で 5% の性能低下，BT で 2% の性能向上と，hostname とほぼ同程度の性能を得られた．より台数が多くなるとこの差はより顕著になり，random は 121 ホストの SP で 30%，BT で 19% 性能が低下した．それに対して ours は SP で 5%，BT で 7% と，それほど性能が低下していない．このように，最大６ホップの単一のクラスタでもこのように性能差が見られた．このアプリケーションでは 100 ホスト以上では台数効果が得られなかったが，より大

規模な問題をより広域な環境で計算した場合に，ネットワークを考慮した Rank 割り当てによって性能差がより大きくなると予想される．我々の推定手法によるトポロジーを用いることで，近いホストが近いホスト名を持つなどと仮定することなく，一般的なアプリケーションに対して通信の効率化を図ることができる．



(a) SP（クラス C）



(b) BT（クラス C）

図 6.5: MPI アプリケーションの Rank 割り当てによる性能比較

第7章　結論

7.1　本論文のまとめ

本論文では効率的なトポロジー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.

7.1.1　トポロジー推定

我々の手法はネットワーク負荷や CPU 負荷が低く，ヘテロなネットワーク環境で短時間にネットワークトポロジーを推定することを目的とし，必要なホスト間の RTT の測定だけを用いてトポロジーを推定する手法を提案した.

実験の結果 1 クラスタ 64 ホストの推定にかかった時間は 4 秒程度，4 クラスタ 256 ホストの推定にかかった時間は 15 秒程度であった．我々の推定アルゴリズムは測定時間について高いスケーラビリティと，遠いホストとの測定をしないという意味で高い局所性を持つことが示された．推定して得られたトポロジー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をグループ分けすることができた．ただし，大きい遅延のリンクに隣接する小さい遅延のリンクを遅延の測定によって推定することは困難であることもわかった.

7.1.2　トポロジーを用いたアプリケーションの効率化

推定したトポロジーを用いて通信効率化の例として 4 つのアプリケーションを示した.

- バンド幅ツリー構築
  トポロジーを用いてリンクを共有しない通信路を導き，並列に測定を行うことによって高速な測定を実現した．実験では 4 クラスタ 256 ホストのバンド幅ツリーの構築にかかった時間は 27 秒程度であった．得られたバンド幅ツリーからホスト間のバンド幅を求めたところ，LAN 内については正しいバンド幅を得られた．一方 WAN では遅延とウィンドウサイズの関係から実際より大きい値を推定してしまうことが多かった.

- 長いメッセージのブロードキャスト
  ツリー上のネットワークでは，ツリーの深さ優先順にメッセージを転送することで最大のバン

ド幅を得られる．実験では 1 クラスタ 64 ホストで実際のネットワークツリーを用いた手法の 88%の性能を得られた．またランダム順に転送した場合は 23%程度しか性能を得られず，スイッチ間のリンクを多数回使用していることを確認した．

- MPI アプリケーションの Rank 割り当て
  MPI アプリケーションの Rank 割り当ては，ホスト名順に行われることが多い．これは近いホスト名を持つホスト同士が近い位置にあるという仮定と，近い Rank のプロセス同士が頻繁に通信するという仮定にもとづく．しかし一般的な MPI アプリケーションの通信最適化のためには，通信パターンとホスト間の通信コストを見積もり，通信量が少なくなる割り当てを行う必要がある．そこで，本論文では実際に近いホスト同士を近くの Rank に割り当てるために，推定したトポロジー上の深さ優先順に割り当てるという手法を近いホスト同士が頻繁に通信を行うアプリケーションに適用した．実験の結果，最大 6 ホップのクラスタ内でも，Rank 割り当てによって 81 ホストで 13%，121 ホストで 30%程度変動することを確かめられた．そして，推定したトポロジーを用いた場合はホスト名順に割り当てた場合と同程度の性能を得られた．このことから，推定したトポロジーを用いてネットワーク的な距離を求めることで MPI アプリケーションの通信最適化を図れることが分かった．

7.2 今後の課題

まず，分岐位置の区間を決定するよりよい方法を考案し，推定精度の向上を目指す．そして，遠いホスト同士は独立に追加することが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で，推定を高速化することを目指す．また，バンド幅ツリー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ーへの適応を目指す．

参考文献

[1] Oliver Aumage and Guillaume Mercier. MPICH/MADIII: a Cluster of Clusters Enabled MPI Implementation. *3rd International Symposium on Cluster Computing and the Grid*, pages 26–33, 2003.

[2] Richard Black, Austin Donnelly, and Cédric Fournet. Ethernet Topology Discovery without Network Assistance. In *ICNP*, pages 328–339, 2004.

[3] Yuri Breitbart, Minos Garofalakis, Ben Jai, Cliff Martin, Rajeev Rastogi, and Avi Silberschatz. Topology discovery in heterogeneous IP networks: the NetInventory system. *IEEE/ACM Trans. Netw.*, 12(3):401–414, 2004.

[4] John W. Byers, Azer Bstavros, and Khaled A. Harfoush. Inference and Labeling of Metric-Induced Network Topologies. *IEEE Trans. Parallel Distrib. Syst.*, 16(11):1053–1065, 2005.

[5] Mark Coates, Rui Castro, Robert Nowak, Manik Gadhiok, Ryan King, and Yolanda Tsang. Maximum likelihood network topology identification from edge-based unicast measurements. *SIGMETRICS Perform. Eval. Rev.*, 30(1):11–20, 2002.

[6] Mathijs den Burger, Thilo Kielmann, and Henri E. Bal. "TOPOMON": A Monitoring Tool for Grid Network Topology. In *ICCS '02: Proceedings of the International Conference on Computational Science-Part II*, pages 558–567, London, UK, 2002. Springer-Verlag.

[7] Mathijs den Burger, Thilo Kielmann, and Henri E. Bal. Balanced Multicasting: High-throughput Communication for Grid Applications. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005.

[8] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Efficient algorithms for large-scale topology discovery. *SIGMETRICS Perform. Eval. Rev.*, 33(1):327–338, 2005.

[9] N.G. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley. Multicast topology inference from measured end-to-end loss. *IEEE Transactions in Information Theory*, 48:26–45, 2002.

[10] Edgar Gabriel, Michael Resch, Thomas Beisel, and Rainer Keller. Distributed Computing in a Heterogeneous Computing Environment. In *Proceedings of the 5th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 180–187, London, UK, 1998. Springer-Verlag.

[11] Arijit Ganguly, Abhishek Agrawal, P. Oscar Boykin, and Renato Figueiredo. WOW: Self-organizing wide area overlay networks of virtual workstations. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 40–41, 2006.

[12] Toshiyuki Imamura, Yuichi Tsujita, Hiroshi Koide, and Hiroshi Takemiya. An Architecture of Stampi: MPI Library on a Cluster of Parallel Computers. In *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 200–207, London, UK, 2000. Springer-Verlag.

[13] Nicholas T. Karonis, Brian Toonen, and Ian Foster. MPICH-G2: a Grid-enabled implementation of the Message Passing Interface. *J. Parallel Distrib. Comput.*, 63(5):551–563, 2003.

[14] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. In *PPoPP '99: Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 131–140, New York, NY, USA, 1999. ACM Press.

[15] Bruce Lowekamp, David O'Hallaron, and Thomas Gross. Topology discovery for large ethernet networks. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 237–248, New York, NY, USA, 2001. ACM Press.

[16] Bruce B. Lowekamp, Brian Tierney, Les Cottrell, Richard Hughes-Jones, Thilo Kielmann, and Martin Swany. Enabling Network Measurement Portability Through a Hierarchy of Characteristics. In *GRID '03: Proceedings of the Fourth International Workshop on Grid Computing*, pages 68–75, Washington, DC, USA, 2003. IEEE Computer Society.

[17] Gary Shao, Fran Berman, and Rich Wolski. Using Effective Network Views to Promote Distributed Application Performance. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 2649–2656, 1999.

[18] Skitter. http://www.caida.org/tools/measurement/skitter.

[19] Kenjiro Taura. GXP: An Interactive Shell for the Grid Environment. In *IWIA '04: Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'04)*, pages 59–67, Washington, DC, USA, 2004. IEEE Computer Society.

[20] Rich Wolski, Neil T. Spring, , and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems, 15(5-6)*, pages 757–768, Oct 1999.

[21] 金田 憲二, 田浦 健次朗, 米澤 明憲. 接続を動的に制御するメッセージパッシングシステム. *Proceedings of 17th Summer United Workshops of Parallel, Distributed and Cooperative Proceeding (SWoPP'04)*, 2004.

発表文献

1. Takayoshi Shiraki, Hideo Saito, Yoshikazu Kamoshida, Katsuhiko Ishiguro, Ryo Fukano, Tatsuya Shirai, Kenjiro Taura, Mihoko Otake, Tomomasa Sato and Nobuyuki Otsu. Real-Time Motion Recognition Using CHLAC Features and Cluster Computing. In Proceedings of the 3rd IFIP International Conference on Network and Parallel Computing, pp.50-56, Tokyo, October 2006.

2. 白井 達也, 田浦 健次朗, 近山 隆. 並列処理のための効率的なトポロジ推定. 2006 年並列／分散／協調処理に関するサマー・ワークショップ (SWoPP2006), pp. 163-168, May 2006.

3. 大武 美保子, 金田 憲二, 鴨志田 良和, 深野 亮, 白木 孝義, 伊藤 聡, 石黒 勝彦, 白井 達也, 斎藤 秀雄, 堀田 勇樹, 南里 卓也, 下畠 康幸, 吉本 晴洋, 酒向 慎二, 杉 正夫, 小谷 潔, 米田 隆一, 林 淳哉, 野口 博史, 田浦 健次朗, 大津 展之, 佐藤 知正. 100 時間ワークショップによる融合教育研究プラットフォームの開発, 日本機械学会ロボティクスメカトロニクス講演会'06 講演論文集, 2A1-B34, May 2006.

4. 白井 達也, 田浦 健次朗, 近山 隆. RTT を用いたネットワークトポロジの推定. 先進的計算基盤システムシンポジウム (SACSIS2006), pp. 267-268, May 2006.

5. 鴨志田良和, 田浦 健次朗, 白井 達也, 斎藤 秀雄, 白木 孝義, 石黒 勝彦, 深野 亮, 大武 美保子, 佐藤 知正, 大津 展之. 冗長性を用いた低遅延並列実時間動作認識システム, 先進的計算基盤システムシンポジウム (SACSIS 2006), pp. 533-540, May 2006.

6. 白井 達也, 斎藤 秀雄, 吉本 晴洋, 鴨志田 良和, 白木 孝義, 石黒 勝彦, 深野 亮, 大武 美保子, 佐藤 知正, 田浦 健次朗, 大津 展之. CHLAC 特徴とグリッドコンピューティングを併用した実時間動作認識, インタラクション 2006 論文集, pp. 97-98, 2006.

7. 白木 孝義, 石黒 勝彦, 深野 亮, 鴨志田 良和, 白井 達也, 斎藤 秀雄, 田浦 健次朗, 大武 美保子, 佐藤 知正, 大津 展之. CHLAC 特徴と Grid コンピューティングを併用したリアルタイム動作認識, 電子情報通信学会技術研究報告, Vol.105, No.615, pp. 97-102, February 2006.

8. 白井 達也, 遠藤 敏夫, 田浦 健次朗, 近山 隆. 高いヒープ使用率の下で高速なインクリメンタル GC. In IPSJ Transactions on Programming. Vol.47 No.SIG 2 (PRO28), pp. 74-83,

February 2006.

9. 白井 達也, 遠藤 敏夫, 田浦 健次朗, 近山 隆. 高いヒープ使用率のもとで高速なインクリメンタル GC. 2005 年並列／分散／協調処理に関するサマー・ワークショップ (SWoPP2005), May 2005.

10. 白井 達也, 遠藤 敏夫, 田浦 健次朗, 近山 隆. 高いヒープ使用率のもとで高速なインクリメンタル GC. 先進的計算基盤システムシンポジウム (SACSIS2005), pp. 226-227, May 2005.

謝辞

本研究を進めるに当たって，近山隆教授，ならびに田浦健次朗助教授には大変貴重なご指摘，ご助言をいただきました．近山隆教授には，要所で的確なアドバイスを頂き，大変お世話になりました．田浦健次朗助教授には研究の方針からプログラムの詳細に至るまで熱心にご指導をいただき，幾度も相談に乗っていただきました．また論文の論理構成や発表方法など，研究者として必要な技術についても数多くのご指導をいただきました．．心より感謝申し上げます．

助手の横山大作さんをはじめ，鴨志田良和さん，斎藤秀雄さんなどサーバを管理をしてくださった方々のお陰で，これほど多数のホスト群を用いた実験を行うことができました．また斎藤さんには，通信アプリケーションの実装の基本的なことから性能向上のテクニックに至るまでさまざまなことを教えていただきました．

研究に行き詰ったときには，高橋慧君，関谷岳史君をはじめ研究室の多くの方に愚痴を聞いていただき，励ましていただき，そして狭まった視野を広げていただきました．ここまで研究を続けられたのも，ひとえに皆様のお陰です．

ほんとうにありがとうございました．

平成 19 年 2 月 2 日