P2P 手法によるインターネットノードの階層的クラスタリング

上田達也[†,†††] 安倍広多[†,††]
石橋勇人[†,††] 松浦敏雄[†,††]

インターネット上のノードを距離に基づいてクラスタリングすることができると，様々なネットワークアプリケーションで有用である．本論文では，インターネット上のノード集合を P2P 方式を用いて階層的にクラスタリングする手法を提案する．既存の手法と異なり，本手法はインターネットの構造に関する外部からの情報を必要とせず，ノード間の距離が測定できればクラスタリング可能であるため，実用性が高い．またシミュレーション実験によって，信頼性・スケーラビリティが高いこと，妥当なクラスタリング結果が得られることを確認している．

Decentralized Hierarchical Internet Hosts Clustering

Tatsuya Ueda,[†,†††] Kota Abe,[†,††] Hayato Ishibashi[†,††]
and Toshio Matsuura[†,††]

Clustering Internet hosts by their network distance is quite useful for many Internet applications. In this paper, we propose a new peer-to-peer algorithm which forms hierarchical clusters of Internet hosts. Our clustering algorithm, which only requires measurability of network distance between any two hosts, is more practical than any other previously proposed one, which requires external information of the underlying Internet structure. In addition, we show simulation results demonstrating that reliability and scalability of our method are high and that our method can generate proper clustering results.

1. はじめに

インターネット上のノード集合をノード間距離に応じてクラスタリング（分割）できると，様々なアプリケーションで性能やスケーラビリティを向上させることができる．大量のトラフィックを多数のノード間でやりとりするようなアプリケーションに対しては特に有効である．

たとえば，P2P オーバレイネットワークでは，オーバレイ上の隣接ノードを，実ネットワーク上で近いノードから選択することにより効率的なルーティングが可能である．アプリケーションレベルマルチキャストでは，マルチキャストツリーをクラスタを用いて構築することにより，効率の良いマルチキャストを行うことができる．また，Web サービスプロバイダは複製サーバをクラスタごとに配置することによりクライアントとのネットワーク遅延時間やサーバの負荷を削減できる．さらに，グリッドのような分散コンピューティングでは，問題領域を分割して各ノードに計算を依頼するが，隣接領域を近いノードに担当させることによりデータ交換にかかる時間を削減できる．

インターネット上のノードのクラスタリングに関しては，これまでにもいくつかの提案が行われている．これらは，サーバクライアント方式に基づく手法と，P2P 方式（完全な自律分散方式）に基づく手法に分けることができる．サーバクライアント方式[1)~3)] では，サーバが全ノードの情報を収集しクラスタリングを行う．しかし，この方式はサーバに負荷が集中するため，クライアント数に対するスケーラビリティや信頼性の面で問題がある．

一方，サーバを必要としない P2P 方式（Pure P2P 方式）は，スケーラビリティや信頼性の面で有利である．P2P 方式を用いてクラスタリングを行うものとして，TOPLUS[4)] がある．しかし，TOPLUS ではクラスタ階層をクラスタリング開始時に構築してしまうため，それ以降のネットワークトポロジの変化に適応

† 大阪市立大学大学院創造都市研究科
Graduate School for Creative Cities, Osaka City University
†† 大阪市立大学学術情報総合センター
Media Center, Osaka City University
††† 有限会社うえだうえおうぇあ
Ueda Ueo Ware, Inc.

できない．また，TOPLUS ではクラスタリング開始時に，多数の BGP ルータから IP アドレスプレフィックス情報を得る必要があり，煩雑である（5 章参照）．

本論文では，以上の問題を解決する手法を提案する．提案手法では，P2P 方式により，インターネット上のノード集合をネットワーク的な距離に基づいてクラスタリングする．距離の単位には IP パケットのホップ数や，RTT（Round Trip Time）等のような，現在のインターネット上で一般ユーザでも容易に測定可能なものを使用する．本手法では BGP ルータから得るような外部情報は不要であり，ノード間の距離を測定するだけでクラスタリング可能であるため，現実のインターネットで利用することはきわめて容易である．さらに，シミュレーションによって，提案手法がノード数に対してスケーラブルであること，生成されるクラスタリング結果が妥当であることを示す．

2. 提案手法の概要

ここでは，提案手法の設計方針と，採用したデータ構造等について述べる．

2.1 設 計 方 針

提案手法は以下のような方針で設計した．

階層的クラスタ P2P による ファイル配布アプリケーション☆やファイル共有サービス，Web コンテンツの分散キャッシュのようなシステムでは，各ノードは自分自身になるべく近いノードからオブジェクトを取得できることが望ましい．たとえば，まず自組織内を探し，存在しなかったら距離が近い組織，さらに存在しなかったらもう少し広い範囲の組織，といったように検索の範囲を順次広げていくことができると都合が良い．このため，クラスタは階層的に構成する．ノードはそれぞれの階層（レベル）で 1 つのクラスタに所属する．また，この構造はスケーラビリティを実現するうえでも役立つ．

動的クラスタリング 1 章であげたようなアプリケーションにクラスタリングを適用することを想定した場合，クラスタリングすべきノード集合はあらかじめ分からない場合が多い．このため，クラスタリングは動的に行う．すなわち，クラスタリングを行った後に，新たなノードが参加したり，すでにクラスタに参加しているノードが離脱したりすることは任意の契機で可能とする．また，ノードは必要に応じて所属クラスタを変更する．

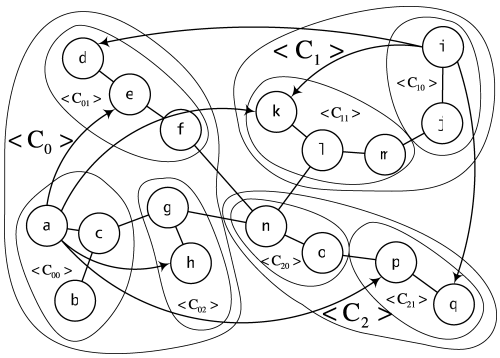
外部情報非依存 BGP ルータから得られるような外部情報を使用しない．ノード間のネットワーク距離の情報だけを使用してクラスタリングを行う．

スケーラビリティ ノード数に対してスケーラビリティが高いこと．

耐故障性・高信頼性 クラスタリングに参加しているノードがどのようなタイミングで故障，あるいはクラスタリングから離脱しても動作を継続できること．

負荷分散 クラスタリングを行ううえで，特定のノードに負荷が集中しないこと．

☆ たとえば，断片化されたファイルを複数のノードから同時にダウンロードすることで，ダウンロードの高速化や負荷分散を行う BitTorrent [5] のようなアプリケーションがこれにあたる．



最上位クラスタを <C0>，<C1>，...
<Ci>のサブクラスタを<Ci0>, <Ci1>, ...と表記している

図 1 クラスタへのポインタの持ち方
Fig. 1 Pointer to clusters.

2.2 クラスタのデータ構造

自分が所属するクラスタと同一の親クラスタに属する他のクラスタ（ただし最上位クラスタの場合は他の最上位クラスタすべて）を**兄弟クラスタ**と呼ぶ．

すべてのノードが全クラスタの情報を持つとデータ量が多くなるため，各ノードは自分が所属する末端クラスタの全ノードへのポインタと，各レベルにおいて各兄弟クラスタに属する**任意の 1 ノード**へのポインタのみを持つ．後者のポインタで指されるノードを**代表ノード**と呼ぶ．代表ノードはクラスタごとに固定されているわけではなく，同一のクラスタを参照する場合にも，参照するノードごとに異なっていてもよい．さらに，自分が所属するクラスタの代表ノードとして自分自身へのポインタを持つ．**図 1** の具体例で説明する．

ノード a は，最上位レベル（レベル 1）ではクラスタ <C0> に属しているため，兄弟クラスタである <C1>，<C2> のノード（ここではそれぞれ k，p）を代表ノードとしている（<C0> に対しては a 自身が代表ノー

ドである）．また次のレベル（レベル 2）では <C_{00}> に属しているため，兄弟クラスタ <C_{01}>，<C_{02}> のノード（それぞれ e，h）へのポインタを持つ．また，図には示していないが，末端レベル（<C_{00}>）のクラスタに対しては，全ノード（ここでは b，c）へのポインタも持っている．他のノードも，すべてこの考え方をもとにポインタを保持する（図では他にノード i のポインタのみ示している）．

<C_2> に対して，ノード a はノード p を，ノード i はノード q を，それぞれ代表ノードとしている．このように，クラスタの特定のノードが代表ノードというわけではなく，すべてのノードは同時に代表ノードになることができる．また，ノード k は，ノード a からは <C_1> の代表ノードとして，ノード i からは <C_{11}> の代表ノードとして指されている．このように，ノードは同時に異なったレベルのクラスタの代表として指されることもある．

なお図 1 の例ではトポロジ上の全ノードがクラスタリングに参加しているが，実際には一部のノードだけが参加しているのが普通である．

2.3 クラスタあたりのサブクラスタ数と階層数

階層的クラスタはツリー構造であるため，ノードが，自分自身が所属するクラスタを選択するコストを抑えるためには，クラスタがバランスしている（1 クラスタあたりのサブクラスタ数が均一である）ことが望ましい．このため，クラスタあたりのサブクラスタ数が基本的に一定値（k）となるようにする．

また，（たとえばノードの粗密に応じて）サブクラスタの階層の深さを可変とすることも考えられるが，P2P システムではノード間で階層の深さに関する合意をとることが困難であるため，階層の深さは固定とする（以下，階層の深さを d とする）．

これにより，ノードは基本的に最大 k^d 個の末端クラスタに振り分けられることになる．k や d の具体的な値については 4.2 節で検討する．

3. 提案手法の詳細

ここでは，提案手法の詳細を述べる．まず 3.1 節で，前述したデータ構造をどのように持つかを述べ，3.2 節以降は動作について述べる．

3.1 データ構造

提案手法で用いるデータ構造を以下に述べる．

3.1.1 クラスタ表

個々のノードは，2.2 節で述べたクラスタ構造を**クラスタ表**で管理する．クラスタ表の各セルには，対応するクラスタの情報（クラスタ情報，3.1.2 項参照）を保持する．例として，図 1 のノード a が持つクラスタ表を**表 1** に記す．

表 1 ノード a のクラスタ表
Table 1 Cluster table of node a.

階層	所属クラスタ	兄弟クラスタ 1	兄弟クラスタ 2
1	<C_0>（のクラスタ情報）	<C_1>	<C_2>
2	<C_{00}>	<C_{01}>	<C_{02}>

表 2 クラスタ情報
Table 2 Cluster information.

クラスタの種類		保持する情報
所属クラスタ	末端以外	クラスタ ID，遠方ノードのリスト，代表ノード（自分自身）
	末端	クラスタ ID，所属する全ノードのリスト，遠方ノードのリスト，代表ノード（自分自身）
兄弟クラスタ		クラスタ ID，代表ノード，バックアップノードのリスト

3.1.2 クラスタ情報

クラスタ情報はクラスタの種類によって内容が異なる．詳細を**表 2** に示す．

それぞれのクラスタには，固有の識別子（クラスタ ID）を付与する．クラスタ ID は，$(random, level)$ で構成される．$random$ は互いに衝突しないように十分なビット数を用いて生成した乱数であり，$level$ はそのクラスタのレベル（1〜d）を表す．

遠方ノードのリストは，当該クラスタに所属するノードのうち，自分との距離を測定済みのものを，距離の遠い順に並べたものである．詳しくは 3.4.4 項で述べる．

バックアップノードは，代表ノードが使用できない場合に代わりに使用するノードである．詳しくは 3.7 節で述べる．

所属ノード数は，当該クラスタに所属するノードの数を表す．この値を定期的に当該クラスタの代表ノードと交換することで，クラスタに属するノード数の概数を得ることができる．

3.1.3 ノード情報

クラスタ情報に記載されている各ノードについて，それぞれ以下の情報（ノード情報）を保持する．

- ノード ID（ノードごとに固有の識別子．乱数によって生成され，ノード無効の通知（3.8 節）等で使用される）
- ノードアドレス（IP アドレス等）
- 最終通信時刻（タイムスタンプ）（3.7 節で述べる）
- 自ノードからの距離（測定してあるノードのみ）

表 3 ノード間メッセージ
Table 3 Inter-node messages.

名前	機能
getTable (宛先クラスタ ID, 発信元クラスタ ID)	クラスタ表を要求する.
notify (宛先クラスタ ID, 新クラスタ ID, 代表ノードの情報)	新クラスタ生成および新ノード参加の通知.
invalidate (宛先クラスタ ID, 削除元クラスタ ID, ノード識別子)	クラスタ表の指定したクラスタから指定したノードを削除する.

- ノード情報の入手元ノードのアドレス（3.8 節で述べる）

3.2 ノード間の通信

ノード間では，**表 3** のメッセージを交換する．各メッセージの使用法については後述する．引数の最初にある宛先クラスタ ID は，どのクラスタに宛てたメッセージなのかを示すために用いる（詳しくは 3.7.1 項で述べる）．

3.3 クラスタリングの初期状態

クラスタリングは，最上位から最下位まで各レベルにクラスタが 1 つだけ存在し，唯一のノード（初期ノード）がそれら全クラスタに所属している状態から開始する．

3.4 クラスタリングへの参加

ここでは，ノードがクラスタリングに参加する際のアルゴリズムを述べる．

クラスタリングに参加しようとするノード（N）は，他の P2P システムと同様，何らかの手段ですでにクラスタリングに参加しているノードを 1 つ知っている必要がある．このノードを N_{ref} とする．

ノードの参加では，次のアルゴリズム（*Join*）を用いて最上位クラスタから順に自分が所属するクラスタを選択していく．この過程で新たなクラスタを生成することもある．以下，レベル i で N が所属するクラスタを $\boldsymbol{C_i(N)}$ と表記する．

3.4.1 アルゴリズム *Join*

(1) $i = 1$ とする．
(2) N_{ref} からクラスタ表を入手する（表 3 の **getTable** メッセージ）．
(3) 入手したクラスタ表を参照し，レベル i クラスタの数が k 未満ならば，3.4.3 項で述べるアルゴリズムでそこに新たにクラスタを生成する．生成したクラスタを $C_i(N)$ とし，探索を終了する．
(4) そうでなければ，3.4.2 項で述べる方法ですべてのレベル i クラスタと N との**近隣度**を計算する．最も近隣度が小さいクラスタが，$C_i(N)$ になる．
(5) ここで，N のクラスタ表のレベル i 行を作成する．負荷分散のため，兄弟クラスタの代表ノードは，入手したクラスタ表に含まれる代表ノードとバックアップノードの中から 1 つを乱数により選択する．残りはバックアップノードとして登録する．
(6) **$i = d$ の場合：**末端クラスタまで到達したので探索は終了する．探索が終了すると，N は末端クラスタ $C_d(N)$ 内の全ノードに表 3 の **notify** メッセージを送ることで，N の参加を通知する．
$i < d$ の場合：$C_i(N)$ の代表ノードを新たに N_{ref} とし，$i = i + 1$ として，(2) に戻る．

なお，トラフィック削減のため，一度得たクラスタ表はキャッシュし，同じノードからはクラスタ表を入手しないようにする．キャッシュは *Join* が終了した時点でクリアする．

3.4.2 近隣度の計算

ノード N_1 と N_2 とのネットワーク上の距離を $\mathrm{dist}(N_1, N_2)$ と表記する☆．

ノード N が複数のクラスタ候補から所属すべきクラスタを選ぶ際，(i) クラスタ内で N に最も近いノード N_c との距離 $\mathrm{dist}(N, N_c)$ が近いクラスタを優先する．また，(ii) $\mathrm{dist}(N, N_c)$ が同じ程度なら，クラスタの中で N から最も離れたノード N_f との距離 $\mathrm{dist}(N, N_f)$ が近いクラスタを優先する．今回はこの条件を数値化するため，N とクラスタとの近隣度を

$$\alpha \cdot \mathrm{dist}(N, N_c) + \mathrm{dist}(N, N_f) \quad (1)$$

と定義した（近い方が値は小さくなる）．なお，α は (i) を (ii) よりもどの程度優先するかを決めるパラメータである．(i) より (ii) を優先するために，α を 1.0 以上に決める．

ノード N とクラスタ C との近隣度を求めるアルゴリズムを**図 2** に示す☆☆．最も近いノードと最も遠い

☆ 距離の尺度としてはホップ数や RTT 等が考えられる．ホップ数を用いる場合は IP ヘッダの TTL フィールドを参照することで，RTT を用いる場合はノード間のパケット往復時間を計測することで，いずれも容易に測定可能である．両側からの計測値が異なる場合は平均をとる等の方法で 1 つの値を決める．

☆☆ クラスタ C が末端レベル（$i = d$）のクラスタの場合，サブクラスタは存在しないので，処理 1 において，C のサブクラスタ集合の代わりにすべての所属ノードとの距離を測定し，処理 2 は省略する．また，すべての距離測定処理では相手ノードの所属クラスタ ID を確認し，想定している ID と異なっていた場合はバックアップノードを用いてリトライを行う（3.7 節参照）．

```
EvalCluster(Cluster C) {
  # 処理 1
  foreach C_x in C のサブクラスタ集合
    C_x の代表ノードと距離を測定する
  end
  上の処理で最も近かったクラスタを C_t とする
  C_t の代表ノードからクラスタ表を入手する

  # 処理 2
  foreach C_y in C_t のサブクラスタ集合
    C_y の代表ノードと距離を測定する
  end
  上の処理で最も近かったクラスタの代表ノード
を N_c とする

  # 処理 3
  N_c のクラスタ表に記載されている C の
  遠方ノードを N_f とする ⋆
  N_f との距離を測定する
  式 (1) によって N と C の近隣度を計算する.
}
```

⋆) N に近いノード N_c から見て遠方にあるノードは，N から見てもやはり遠方にあると仮定している．

図 2 近隣度を求めるアルゴリズム
Fig. 2 Neighborness calculation algorithm.

ノードを正確に求めるにはコストがかかるので，図 2 で述べる手法では近似的に求めている．

3.4.3 クラスタの生成

Join で，ノード N が新たにレベル i クラスタ C を生成する場合，次のように行う．

(1) N は，レベル i から d まで，N が所属するクラスタの ID を生成し，N のクラスタ表に登録する．

(2) C の兄弟クラスタに属するノードは，C を自身のクラスタ表に登録する必要がある．このため，これらのノードに，C のクラスタ ID と，その代表としての N に関する情報（ノード識別子や IP アドレス等）を通知する（C のサブクラスタの存在は N 以外のノードは知らなくてよいため，通知する必要はない）．
N は *Join* の実行により C のすべての兄弟クラスタの代表ノードを知っているので，まずそれらに通知する（表 3 の **notify** メッセージ）．通知を受けた代表ノード（N_r）は，そのクラスタ（C）を自分のクラスタ表に加えると同時に，自分の知るサブクラスタの代表ノードに（再帰的に）通知する．また，これと同時に $C_d(N_r)$ に所属する全ノードにも通知を行う．

このアルゴリズムでは，あるクラスタの生成が関連ノードに伝播する前に，別のノードが同一のクラスタの下に別のクラスタを生成する可能性がある（**クラスタ生成の衝突**）．衝突が発生すると k より多くのクラスタが生成される可能性がある．衝突の可能性は伝播時間が長いほど大きくなるが，クラスタの生成は最上位レベルから順番に行われるため，クラスタ生成を伝播させる必要があるノードの数は通常 k 未満にすぎず，実際には衝突の可能性は低い．

クラスタ生成の衝突が発生した場合，新たに生成されたクラスタどうしは相手の存在を認識できないが，この状況の解消方法は 3.11 節で述べる．

3.4.4 遠方ノードの収集

Join や *Rejoin*（3.6.2 項参照）等の実行により，ノード N がクラスタ C に属するノードと距離を測定した場合，N のクラスタ表に含まれる C のエントリの遠方ノードに当該ノードを登録する．遠方ノードリストは遠い順にソートし，一定数だけ保持する．

3.4.5 最 適 化

Join では，各レベルですべてのクラスタとノード N との近隣度を計算する．その際，次の方法で距離の測定回数を削減する．

- 各クラスタの代表ノードとの距離を測定し，クラスタを距離の昇順に並べたリストをつくる．
- リストの先頭のクラスタを C_{best} とする．C_{best} と N の近隣度を 3.4.2 項のアルゴリズムで計算し，その値を P_{best} とする．
- 残りのクラスタについて，順次以下の処理を行う．
- クラスタ C で，$\mathrm{dist}(N, N_c) = min$，$\mathrm{dist}(N, N_f)$ = 代表ノードとの距離，として仮の近隣度を計算する．ここで，min は，使用する距離尺度での距離の最小値である．たとえばホップ数を使用する場合，$min = 1$ とする．
- （実際の近隣度 $\geq$ 仮の近隣度）が成り立つので，仮の近隣度が，P_{best} よりも大きければ，C を所属先として選択することはない．このため，C との近隣度の計算は省略する．
- 仮の近隣度が P_{best} より小さければ，実際に C との近隣度 P_C を計算する．P_C が P_{best} より小さければ $C_{best} = C$，$P_{best} = P_C$ とする．

3.5 クラスタリングからの離脱

ノードは自発的，あるいは突発的にクラスタから離脱する可能性がある．本アルゴリズムではいずれの場合も同様に扱い，他ノードへの通知等の処理は行わない．

3.6 ノードの引越し

クラスタに新たなノードが参加したり，新しいクラスタが生成されたりするにつれ，すでに参加しているノードの所属するクラスタが必ずしも最も近隣度の小さいものではなくなっている場合がある．また，*Join* においてクラスタの近隣度を計算する際，クラスタ内で最も自分に近いノード（N_c）等を近似的に求めているため，最初の *Join* では必ずしも最適なクラスタに所属しないこともありうる．

このため，(1) より近いノードを兄弟クラスタ内で発見した場合に代表ノードを切り替える．また，(2) 定期的に自分の所属クラスタを選択しなおすアルゴリズム（*Rejoin*）を実行し，より近隣度の小さいクラスタを見つけた場合は所属クラスタを変更する（ノードの引越し）．以下で詳しく述べる．

3.6.1 より近いノードへの代表ノード切替え

ノード N がクラスタ C の近隣度を計算するうえで，N に近い C のノード（N_c）をなるべく正確に求めることは重要である．3.4.2 項のアルゴリズムでは N_c を近似的にしか求めていないため，次の方法で，より自分に近いノードを発見した場合に代表ノードを変更する．

あるノード N が，あるクラスタ C に所属するノード R から getTable リクエストを受けたときに，N は R との距離を測定し，N が現在保持する C の代表ノードとの距離よりも，R のほうが近ければ，N の C に対する代表ノードを R に変更する☆．

これにより，次の *Rejoin* 実行の際に，クラスタの近隣度をより精度良く求めることが可能となる．

3.6.2 アルゴリズム *Rejoin*

Rejoin は，以下に述べる違いを除いて *Join* と同じである．

- ノード N が，自分が所属するクラスタ $C_i(N)$ との近隣度を計算する際，$N_c = N$ としてしまうと，N の近くに $C_i(N)$ に所属するノードが存在しなくても $C_i(N)$ の近隣度が小さくなってしまう．このため，この場合はクラスタ表上でレベル $i+1$ から d のレベルの兄弟クラスタの代表ノード，および末端クラスタの全ノードで，最も近いノードを N_c とする．
- クラスタ C が，*Rejoin* 前・後の両方のクラスタ表に兄弟クラスタとして存在し，かつその代表ノードが変化しなかった場合，新しい代表ノードは C の代表ノードとバックアップノードから乱数で選びなおす．これは，(1) 様々なノードを代表ノードとして使用したほうが 3.6.1 項の処理に役立つ（より近いノードを発見できる可能性が高まる），(2) 3.6.1 項の処理だけではクラスタに近いノードが集中的に代表ノードとなってしまうが，これを緩和する，という理由からである．
- あるレベルのクラスタ数が k より少ない場合も新しいクラスタを生成しない．この理由は，*Rejoin* は定期的に行われるので，*Join* 時よりもクラスタ生成が衝突する可能性が高いためである．

3.7 代表ノード喪失の防止

ノードが別のクラスタへ引越ししたり，あるいはクラスタリングから離脱した場合，そのノードを代表ノードとしていたノードは，代表ノードを変更する必要がある．

このため，ノードはクラスタに対して，代表ノード以外にそのクラスタに属するノードの情報を最大 $\boldsymbol{b}$ 個分保持しておくことにした（**バックアップノード**）．代表ノードが使用できなくなった場合，バックアップノードを順に代表ノードとして，処理の継続を試みる．以下詳しく述べる．

3.7.1 代表ノード無効の検出

ノードは，次の契機で自身の代表ノードが無効であることを検出する．(1) ノード N が代表ノード D に何らかのリクエストを送信する際，そこには必ず “宛先クラスタ ID” が含まれている（表 3）．リクエストを受信したノード D は，それが自分が現在所属しているクラスタへのリクエストでない場合，ノード N に対してエラーを返す．これにより，ノード N はノード D が目的のクラスタから離脱していることを知ることができる．また，(2) ネットワーク的に代表ノードと通信できない場合は，タイムアウト等で判断する．

3.7.2 バックアップノードの収集

ノード N は，バックアップノードを次のようにして収集する．

- N が，（*Join* や *Rejoin* 等の契機で）クラスタ C の代表ノード N_r から getTable によりクラスタ表 T を得た場合，T 上で C の下位クラスタのエントリに含まれるノード（代表ノードとバックアップノード）は C のバックアップノードとして登録できる．また，N の兄弟クラスタで，T に含まれるものがあれば，そのエントリに含まれるノードも当該兄弟クラスタのバックアップノードとして登録できる．
- N が C の代表ノードとして N_r を指していると

☆ 表 3 の getTable の引数にある発信元クラスタ ID は R が所属するクラスタを伝えるために使用する．

きに，N_r が引越しをしたとする．N が N_r と通信を試みると宛先クラスタ ID の不一致によりエラーとなるが，このとき，N_r は，エラー情報とともに，N_r が引越し前に使用していたクラスタ表（**旧クラスタ表**）も送る．この中には，C の下位クラスタの代表やバックアップノードが記載されているので，N はこれらをバックアップノードに登録する．この中にまだ C に属しているノードが見つかる可能性は高い．

最後に通信してから時間を経ているノードはすでに引越ししている可能性があるので，C のバックアップノードとしては，C に属していることが最近確認できたノードを優先的に確保しておくことが望ましい．このため，各ノードは代表ノードとの通信が発生するたびに，ノードのクラスタ表のタイムスタンプを更新する．クラスタ表を別のノードに送信する際には，タイムスタンプも含めて送信することにより，受信したノードはクラスタごとに最新 b 個のバックアップノードを維持することができる．

3.8 ノード無効の通知

あるノード（N_1）が得る他のノード（N_2）の情報は，さらに別のノード（N_3）から提供されたものである可能性がある．たとえば *Join* では，ノードは N_{ref} から N_c や N_f 等へのポインタを得る．また，ノードはバックアップノードを他のノードから得る（3.7.2 項参照）．しかし，提供される情報は最新のものであるとは限らない（3.7.1 項参照）．この場合，N_1 は N_3 に対して N_2 の無効を通知し，N_3 はクラスタ表および旧クラスタ表から N_2 を削除する（表 3 の **invalidate** メッセージ）．これを実現するため，クラスタ表に登録する他のノードの情報には，その情報の入手元ノードを登録しておく（3.1.3 項参照）．

クラスタリングのすべての処理で，リモートノードが無効だった場合は，リモートノードの入手元ノードに無効を通知する．

3.9 クラスタの削除

クラスタ C に所属している全ノードが離脱したら，C は消滅させる．このためには，C の兄弟クラスタの全ノードのクラスタ表に保持されている C のエントリを削除する必要がある．

今回は，C の兄弟クラスタに所属するノードが C の代表ノードを失い，バックアップノードを使っても回復できなかった場合に，C のエントリを削除するという方法を採用した．ただし，この方法ではまだ所属ノードが残っているクラスタのエントリを誤って削除してしまう可能性がある．この可能性を少なくするためには，十分なバックアップノードを保持しておく必要があるが，このための方策は，3.10 節で述べる．また，誤って削除してしまったクラスタ情報の修復方法は 3.11 節で述べる．

3.10 ノードの広告

あるクラスタ C に対して，C の兄弟クラスタのノードが十分な数のバックアップノードを保持していない場合，C は誤って削除される可能性が高くなる．このため，*Join* や *Rejoin* の結果，$|C_i(N)| \leq b$（$|C_i(N)|$ は $C_i(N)$ の所属ノード数）となるクラスタ（$i = 1 \sim d$）に所属したノード N は，$C_i(N)$ の兄弟クラスタの全ノードに，新しいノードが参加したことを通知する．通知を受けたノードは，当該ノードをバックアップノードとして登録する．

3.11 クラスタ情報の修復

実在する兄弟クラスタ C のエントリをクラスタ表に持たないノード N が存在したとする（3.4.3 項，3.9 節）．この場合，他に 1 つでも C のエントリを持つノードが存在すれば，N は定期的に行う *Rejoin* の過程で，一定時間後には C をエントリに持つクラスタ表を受け取る．N は，C の代表ノードやバックアップノードと通信を試み，成功した場合は，N のクラスタ表に C のエントリを回復する．

4. 評価と考察

提案手法を評価するため，シミュレータを作成し実験を行った．

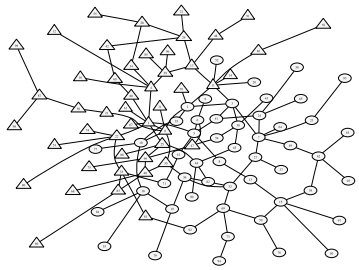
4.1 シミュレータの概要

本シミュレータは与えられたネットワークトポロジに対して本論文で提案するクラスタリングを実行する．4.5 節を除いて，実験で用いたトポロジは乱数を用いて作成した☆．このネットワークトポロジに対して，本シミュレータは各ノードを個別にランダムな順番でクラスタリングに参加させる．

本シミュレータにおいてはノード間の距離尺度としてはホップ数を用いる．ノード間の経路はホップ数最小のものを選択する．ノード間でメッセージを交換することによりノード間のホップ数を求めることができるため，シミュレータ上では図 2 のすべての距離測定は getTable メッセージを用いて行うようにした☆☆．これにより “C_t の代表ノードからクラスタ表を入手

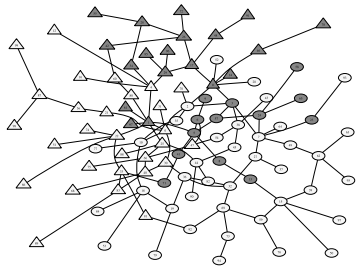
☆ まず乱数を使ってツリー型トポロジを生成する．次にランダムに選んだ 2 つのノード間にリンクを作成する．このリンク作成作業を (全ノード × 0.2) 回繰り返した．

☆☆ getTable を用いる理由は距離測定と同時にバックアップノードの収集を行うことができるためである．



第 1 レベルクラスタ

同じ形をしたノードは同じ第 1 レベルクラスタに属する



第 2 レベルクラスタ

同じ形で同じ色のノードは同じ第 2 レベルクラスタに属する

図 3 100 ノードのネットワークをクラスタリングした例
Fig. 3 An example of clustering result (100 nodes).

する" 処理は（キャッシュがあるため）省略できる．

本シミュレータによるクラスタリング例を**図 3** に示す☆．これは，100 ノードからなるネットワークトポロジを生成し，全ノードをランダムな順序で *Join* させ，さらに各ノードで 1 回だけ *Rejoin* を行った結果の最上位 2 階層の図である．第 1 レベルの 2 つのクラスタが，第 2 レベルではさらにそれぞれ 2 つに分割されている様子が分かる．

4.2 クラスタあたりのサブクラスタ数と階層数に関する考察

クラスタリングに参加するノード数を n とする．今，ノードが参加するクラスタを選択するために必要な距離測定回数を最小とするような，クラスタあたりのサブクラスタ数 k と階層数 d の値を決定することを考える（2.3 節参照）．

全クラスタのサブクラスタ数（末端クラスタの場合は所属ノード数）が k である場合に，あるノードが全階層でクラスタを選択するために必要な距離測定回数は，

$$d(2k^2 - (p+2)k + 1) - (k^2 - 3k + 2 - p) \quad (2)$$

である（付録参照）．ただし p は，N_f が（たまたま）処理 1 あるいは処理 2 で距離測定済みである確率である．k は 1 より大きい整数であることを考慮すると，式 (2) を最小にする k は 2 となるため，クラスタあたりのサブクラスタ数は 2 とするのが適切であるといえる．

このとき必要な階層数は

$$\lceil \log_2 n \rceil - 1 \quad (3)$$

となるが，実際のクラスタリングでは偏りが生じるため，これをクラスタリングの階層数 d としてしまうと，末端クラスタの所属ノード数が k（$= 2$）を大幅に超えてしまう場合がある．近隣度の計算（3.4.2 項参照）やノードの広告（3.10 節参照）には末端クラスタの所属ノード数に比例したコストがかかるため，末端クラスタの所属ノード数は抑える必要がある．

d を式 (3) よりも十分大きく設定すれば，末端クラスタのノード数を抑えることができるため，d を式 (3) からどの程度大きくとるべきかを実験に基づいて検討した．

ノード N に対して，$h(N) = (|C_i(N)| = 1$ となる最小の $i)$ と定義する（ただし，そのような i が存在しない場合は $h(N) = d$ とする）．$h(N)$ は，ノード N に必要十分な階層数を示す．

ノード数 3,000 のトポロジで次のような実験を行った．$n = 3{,}000$ 時の式 (3) の値は 11 であるが，d をそれよりも大きく設定（今回は 30）し，そのうえで全ノードをランダムな順番で *Join* させた．その後，定期的な *Rejoin* をシミュレートするため，全ノードがランダムな順序で *Rejoin* を行う処理を 3 回繰り返した．このとき，*Join* 直後および各々の *Rejoin* 後の $h(N)$ の分布を**図 4** に示す．実験は乱数で生成した 3 つのトポロジ上で行い，平均を求めている．

図より，$h(N)$ はほぼ式 (3) の値（$= 11$）を中心に分布し，*Rejoin* を繰り返すことによってほとんどのノードの $h(N)$ は式 (3) の 2 倍程度に収まることが分かる．このため，d の値は，クラスタリングに参加すると見込まれるノード数から算出した式 (3) の値の 2 倍程度の値を設定すればよいと考えられる☆☆．

4.3 バックアップノード数に関する考察

バックアップノード数の最大値 b を小さくすると，ノードの広告が発生する頻度を減らすことができるが，小さすぎると実際にはまだ存在するクラスタを削除してしまう（3.9 節参照）可能性が高くなる．適切な b を求めるために，いくつかの b の値（0〜4）に

☆ レンダリングには Graphviz [6] を使用している．

☆☆ 式 (2) は d の値に比例して大きくなるが，実際の近隣度計算においては，階層が $h(N)$ に達した時点で近隣度の計算は終了する．

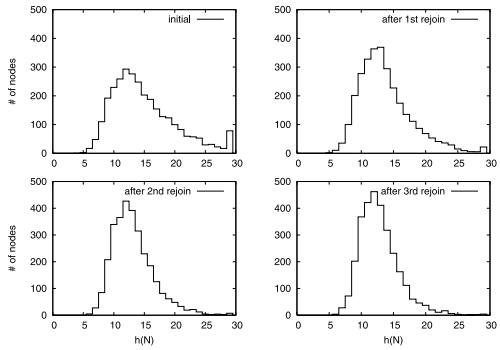


図 4 $h(N)$ の分布（3,000 ノード時）
Fig. 4 Distribution of $h(N)$ (3,000 nodes).

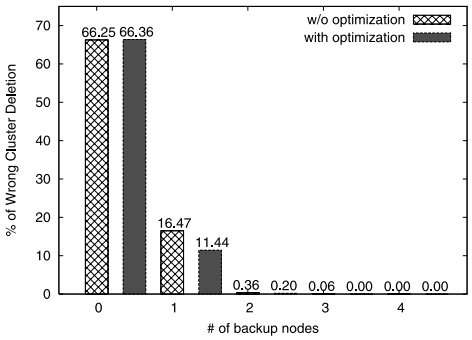


図 5 クラスタを誤って削除する確率（3,000 ノード時）
Fig. 5 Probabilities of wrong cluster deletion (3,000 nodes).

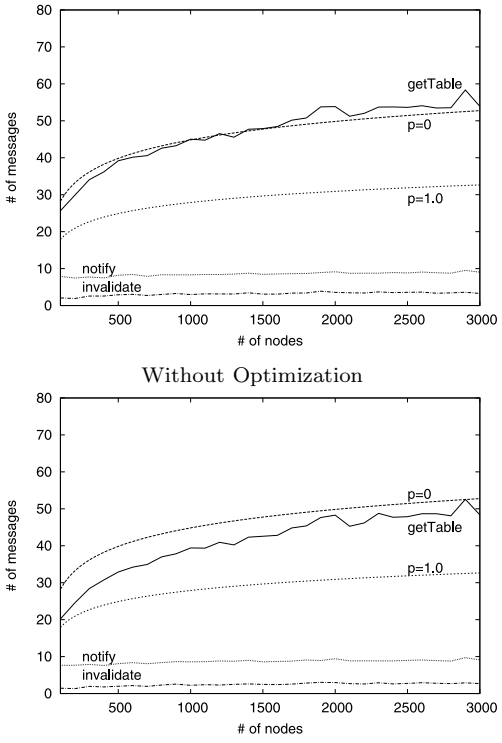


図 6 1 ノードあたりの *Rejoin* に必要なメッセージ数
Fig. 6 Number of message for *Rejoin* (per one node).

対し，存在するクラスタを誤って削除する確率を求めた．誤って削除されるクラスタは所属ノード数の少ない末端階層付近のクラスタであると考えられるが，このようなクラスタは総ノード数の多寡に関係なく同程度の割合で存在する．このため，本実験ではノード数を 3,000 に固定した．乱数で生成した 3 つのトポロジで得られた値の平均値を**図 5** に示す．

この実験では b が少なくとも 4 あればクラスタが誤って削除されることはなかった（なお，万一削除されても 3.11 節の手順で回復することができる）．

4.4 スケーラビリティに関する評価

提案手法のスケーラビリティをノード間で交換されるメッセージ数に基づいて評価した．この実験では，前節までの評価結果に基づきパラメータとして $k = 2$, $d = 30$, $b = 4$ を採用した．またノード数 n は 100 から 3,000 の間で 100 ずつ変化させた．

全ノードが参加した後で，各ノードが *Rejoin* を行うために要したメッセージ数をカウントした（メッセージの種類は表 3 を参照）．*Rejoin* に要するメッセージ数と新規参加に要するメッセージ数はほとんど同じであるため，この数はこの規模のクラスタリングに参加するためのコストと考えてよい．なお，4.1 節に述べたように，距離測定は getTable で行うものとしている．

Rejoin は 5 回行い，1 ノードあたりの平均メッセージ数を求めたものを**図 6** にプロットした（乱数で生成した 3 つのトポロジで測定し，値を平均している）．比較のため，3.4.5 項の最適化を行わない場合と行った場合の両方の結果，および，式 (2) において $d = \lceil \log_2 n \rceil - 1$, $p = \{0, 1\}$ とした結果も示している．

今回の実験により getTable メッセージの数は，ほぼ式 (2) に従うことが確認された．実験結果が $p = 0$ のラインよりも上回る場合があるのは，代表ノードが引越していた場合に，バックアップノードからクラスタ表を再取得している影響と考えられる．また，invalidate，notify メッセージ数はほぼ一定数で推移していること，最適化により getTable メッセージ数が抑えられることも確認できる．

実験では 3,000 ノードまでのクラスタ集合に対して，1 つのノードが参加する際に必要なメッセージ数を計測したが，より多数のノードで構成されるクラスタ集

合に参加する場合について，以下に考察する．

式 (2) および式 (3) から，n 個のノードで構成されているクラスタ集合に，1 ノードが参加するために必要な getTable メッセージの数は

$$(5-2p)(\lceil \log_2 n \rceil - 1) + p \qquad (4)$$

となる．

ここで，式 (4) は $p=0$ のときに最大値となるので，$n = 2^{16} = 65{,}536$ の場合でも，各ノードあたりの getTable メッセージ数はたかだか 75 であると予測できる☆1．さらに，notify，invalidate の各メッセージは実験結果から，ノード数にかかわらずほぼ定数（あわせてたかだか 15 程度）であると予測できるので，数万ノードで構成されるクラスタ集合に，新たにノードが参加する際に必要なメッセージ数は 100 を大きく上回ることはないと推測できる．クラスタリングは，トラフィックを大量に生成するアプリケーションに対して有効であるため，この程度のオーバヘッドは許容範囲であると考えられる．

4.5 クラスタリングの評価

本節では，生成したクラスタの質を評価するために，より現実に近いトポロジとして Internet Topology Generator（Inet-3.0）[7] によって生成されたトポロジを使用する．

4.5.1 対照実験

対照実験として K-means 法と乱数によるクラスタリングを行い，提案手法によるクラスタリング結果と比較した．

K-means 法は統計データをクラスタリングするための一般的なアルゴリズムである．K-means 法を用いてノード集合を K 個のクラスタに分割するには，次の 3 段階の操作を行う．(1) 適当なノードを K 個選び，それぞれをクラスタの中心とする．今回は，ノード集合内で最も遠い 2 点を最初の中心とした．(2) 残りの各ノードは自分に最も近い中心が属するクラスタを所属クラスタとする．全参加ノードが所属クラスタを決定したら，各クラスタの中心ノード☆2を新たな中心とする．(3) クラスタが安定するまで，(2) を繰り返す．

K-means 法では，全ノードが既知であり，全ノード間の距離があらかじめ分かっている必要があるので，インターネット上で実用的に適用することは困難であるが，比較的良いクラスタリング結果が得られるので（このため文献 1) でも使用されている），提案手法がどの程度 K-means 法に近づくことができるかを調べるために導入した．

一方，何も工夫しない場合の例として乱数によるクラスタリングを評価した．距離に基づかず，乱数で所属ノードを決定することで，ノードは均等に各クラスタに分散される．

4.5.2 クラスタリング評価の指標

階層的クラスタリングを評価するために，**Average Traversal Cost for Hierarchical Flooding（ATC）**と呼ぶ指標を考案した．アプリケーションレベルマルチキャストのように，あるノードから他の全ノードへデータを配布する必要がある場合を想定し，あるパケットを全ノードが受け取るために必要なネットワーク的距離の総和を ATC として定義する．ATC が小さいほど，ネットワーク全体にかかる負荷が小さく，良いクラスタであるといえる．

ノード集合が階層的にクラスタリングされている場合，全ノードにデータを配布するには，最上位クラスタの代表ノードにのみパケットを送信すればよい．パケットを受け取ったノードは下位クラスタの代表ノードに対して再帰的にパケットを転送する．

したがって，あるクラスタ C の ATC は次式で示すように再帰的に定義される．

$\mathrm{ATC}(C) = C$ のサブクラスタの ATC の和
$+ C$ のサブクラスタ間平均累積距離☆3

ただし最下層では，すべての所属ノードにパケットを送信するので，サブクラスタ間の平均累積距離の代わりにクラスタ内ノード平均累積距離☆4を加算する．

また，クラスタリングをしない場合の総トラフィック量の期待値と，クラスタリングした場合の ATC の

☆1 実際には代表ノードが引越していた場合，クラスタ表を再取得するため，75 を超えることがあるが，図 6 からその確率は低いと考えられる．

☆2 所属クラスタ内で，最も遠いノードまでの距離が最も小さいノードを中心ノードとする．

☆3 クラスタ C_1 と C_2 のクラスタ間平均距離 $\mathrm{cdist}(C_1, C_2)$ を以下のように定義する．

$$\mathrm{cdist}(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{\forall s \in C_1, \forall t \in C_2} \mathrm{dist}(s,t)$$

あるノードが所属するクラスタ C の全ノードにデータを配信する場合，C の全サブクラスタの代表ノードにデータを送信すればよい．このときに必要なトラフィックの期待値（クラスタ C のサブクラスタ間平均累積距離）は以下のようになる．

$$\frac{1}{|\mathcal{S}|} \sum_{\forall C_1 \in \mathcal{S}, \forall C_2 \in (\mathcal{S} - C_1)} \mathrm{cdist}(C_1, C_2)$$

ただし $\mathcal{S}$ は C の全サブクラスタの集合．

☆4 クラスタ C のクラスタ内ノード平均累積距離は以下のとおり．

$$\frac{1}{|C|} \sum_{\forall s \in C, \forall t \in (C-s)} \mathrm{dist}(s,t)$$

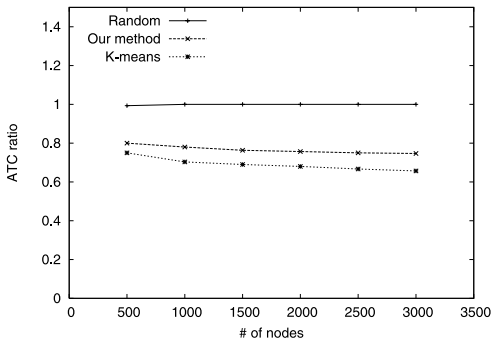


図 7 ATC 比
Fig. 7 ATC ratio.

比を ATC 比と呼ぶ．ATC 比をみることでクラスタリングの効果を把握することができる．

実験では，Inet-3.0 で 6,000 ノードのトポロジを生成した．そのうえで，乱数で選ばれた一定数のノードが (1) K-means 法による，(2) 提案手法による，(3) 乱数による手法でそれぞれにクラスタリングを行い，生成されたクラスタリングを，すべての最上位クラスタを包含する仮想的クラスタの ATC で評価した．

図 7 に ATC 比を示す．参加ノード数は 500 から 3,000 までの間で 500 ずつ増加させ，全ノード数が同一（6,000 ノード）の異なる 3 種類のトポロジを用いて結果を平均した．

乱数によるクラスタリングでは ATC 比が 1.0 となり，クラスタリングの効果がまったくないことが分かる．これに対して提案手法は，(1) クラスタリング開始時点では参加するノードが決まっていないこと，(2) 限られたノード間の距離情報しか使わないこと，という厳しい条件下でクラスタリングを行うことを考慮すれば，K-means 法と比べても遜色のない評価値が得られたといえる．

5. 関 連 研 究

既存のクラスタリング方式を大別すると，サーバクライアント方式と P2P 方式に分類できる．また，クラスタリングを，ノード間の距離に基づいて行うか否かという分類も可能である（**表 4**）．

サーバクライアント方式で，ノード間距離に基づく手法としては文献 1) がある．文献 1) では最初に距離測定のための目印となるノード（ランドマーク）をいくつか選定し，各ノードはランドマークとの距離を測定することでクラスタリングを行う．一方，ノード間距離に基づかない手法としては文献 2)，文献 3) がある．文献 2) では，IP アドレスと BGP のネットワークプレフィックス情報とをあわせてクラスタリングする方法を，文献 3) では同一の DNS サーバを名前解決に使用しているノードを同じクラスタとする方法を提案している．

表 4 クラスタリング方式の分類
Table 4 Taxonomy of clustering algorithms.

	ノード間距離に基づくもの	ノード間距離以外に基づくもの
サーバクライアント方式	文献 1)	文献 2)，3)
P2P 方式	**提案手法**	TOPLUS [4]

しかし，これらのサーバクライアント方式による手法では，特定のノードに負荷が集中するため，ノード数に対するスケーラビリティや信頼性の面で問題がある．

P2P 方式を用いてクラスタリングを行うものとして，TOPLUS [4] がある．TOPLUS は，ノード間距離ではなく，ノードの IP アドレスをベースに階層的なクラスタリングを行い，そのうえでキーの格納・参照サービスを提供する．TOPLUS ではクラスタ階層を，(1) インターネット上で使用されている IP アドレスプレフィックスの情報を利用してあらかじめ構築しておく．あるいは (2) ノードの IP アドレスを適当な位置（8，16，24 ビット目等）で分割することで構築する，という方法が提案されているが，それぞれ次のような問題点をかかえている．

(1) インターネット上で使用されている IP アドレスプレフィックスは日々変化するため，クラスタ集合も変化に適応することが望ましいが，TOPLUS ではクラスタリング開始前にクラスタ集合を生成してしまうため，それ以降の変化には対応できない．また，IP アドレスプレフィックス情報は，BGP ルータやルーティングレジストリから得られる情報を総合することで得るとしている．しかし，ルーティングレジストリにすべての IP アドレスプレフィックス情報が登録されているわけではない．また，BGP ルータでは経路情報の集約（aggregate）を行うため，いくつかの異なるネットワークが 1 つの IP アドレスプレフィックスにまとめられてしまう可能性がある（これは特に国際ゲートウェイで顕著である）．このため，精度良くクラスタリングを行うためにはできるだけ多くの BGP ルータからの情報が必要であるが，現在インターネット上に存在する BGP ルータの多くは情報を一般に公開していない．公開しているルータもあるが，情報を取得する手順は個々のルータによって異なるため（Web の CGI プ

ログラムや telnet インタフェースが用いられることが多い），それぞれのルータに応じて異なった取得方法を用いる必要がある．このため，実際のアプリケーションで BGP ルータからの情報を用いることはそれほど容易ではない．さらに，同一の IP アドレスプレフィックスに属する組織内部（AS 内部等）でのクラスタリングができないという問題もある．

(2) IP アドレスを単純に区切ってクラスタリングを行う手法では，クラスタがトポロジと乖離する場合が多いことが知られている[2]．

これに対して，提案手法ではノード間距離に基づいて P2P 方式によりクラスタリングを行っている．一般ユーザでも自由に入手可能な情報だけを用いてクラスタリングを行うため，現在のインターネット環境で容易に使用できる．また，事前にクラスタ集合を生成するのではなく，必要に応じて動的に生成している点も TOPLUS と異なっている．

一方，アドホックネットワークの分野では，ノード間で交換するルーティング情報を削減するために，直接通信できるノードをクラスタリングすることが行われる[8),9)]．しかし，アドホックネットワークではノードは直接通信できる範囲にある他のノードを容易に知ることができるのに対し，一般のインターネットでは IP アドレスさえ分かればどのノードとも通信できるという違いがある．このため，アドホックネットワークでのクラスタリングアルゴリズムをインターネットでのクラスタリングで使用することはできない．

6. おわりに

本論文では，P2P 手法によって多数のインターネットノードを階層的にクラスタリングする手法を提案した．さらに提案手法のシミュレータを作成し，実験によって，本手法で提案するアルゴリズムを評価した．

本手法は完全な自律分散アルゴリズムであり，BGP ルータ等から供給されるような外部の情報を必要とせず，ノード間の距離さえ測定できればクラスタリングできる．ノード数に対するスケーラビリティは高く，生成される階層クラスタの質も妥当であるため，本手法は様々なネットワークアプリケーションで活用できると考えられる．

今後の課題としては，インターネット上での実験による提案手法の実用性の検証，階層的クラスタを活用した様々なアプリケーションの設計・実装等があげられる．

謝辞 本研究の一部は，日本学術振興会科学研究費補助金基盤研究（C）（2）16500039 の助成により行われた．

参考文献

1) Agrawal, A. and Casanova, H.: Clustering Hosts in P2P and Global Computing Platforms, *Proc. Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, Tokyo, pp.367–373 (2003).
2) Krishnamurthy, B. and Wang, J.: On Network-Aware Clustering of Web Clients, *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, New York, pp.97–110, ACM Press (2000).
3) Bestavros, A. and Mohrotra, S.: DNS-based Internet Client Clustering and Characterization, *Proc. 4th IEEE Workshop on Workload Characterization* (*WWC'01*), Austin, pp.159–168 (2001).
4) Garcés-Erice, L., Ross, K.W., Biersack, E.W., Felber, P.A. and Urvoy-Keller, G.: Topology-centric look-up service, *Proc. COST264/ACM 5th International Workshop on Networked Group Communications* (*NGC*), Munich, Germany (2003).
5) BitTorrent, Inc.: BitTorrent (2005). http://www.bittorrent.com/
6) AT&T Labs: Graphviz — open source graph drawing software (2004). http://www.research.att.com/sw/tools/graphviz/
7) Winick, J. and Jamin, S.: Inet-3.0: Internet Topology Generator, Technical Report CSE-TR-456-02, University of Michigan.
8) Iwata, A., Chiang, C.-C., Pei, G., Gerla, M. and Chen, T.-W.: Scalable Routing Strategies for Ad Hoc Wireless Networks, *Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks*, pp.1369–1379 (1999).
9) Chiang, C.-C., Wu, H.-K., Liu, W. and Gerla, M.: Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel, *Proc. IEEE SICON'97*, pp.197–211 (1997).
10) 上田達也，安倍広多，石橋勇人，松浦敏雄：P2P によるインターネットノードの階層的クラスタリング手法の提案，情報処理学会研究報告，Vol.2004, No.96, 2004-DSM-35, pp.59–64 (2004).
11) 上田達也，安倍広多，石橋勇人，松浦敏雄：P2P によるインターネットノードの階層的クラスタリング手法の評価，マルチメディア，分散，協調とモバイル（DICOMO 2005）シンポジウム論文集，IPSJ Symposium Series, Vol.2005, No.6, pp.77–

表 5 クラスタの近隣度計算に必要な距離測定回数
Table 5 Required number of distance measurement to calculate neighborness of a cluster.

	最上位クラスタ	中間クラスタ	末端クラスタ
処理 1	k	$k-1$ または 0	$k-1$ または 0
処理 2	$k-1$	$k-1$	0
処理 3	$1-p$	$1-p$	0
合計	$2k-p$	$2k-1-p$ または $k-p$	$k-1$ または 0

80 (2005).

付　　録

近隣度計算に要する距離測定回数

全クラスタのサブクラスタ数（末端クラスタの場合は所属ノード数）が k であるとして，あるノードがクラスタリングに参加する際の近隣度計算に要する距離測定回数を考察する．

まず，ある最上位クラスタ（C_1）との間で近隣度を計算する場合を考える．C_1 のサブクラスタは k 個存在するため，処理 1（図 2 参照）で代表ノードとの距離測定は k 回行う．処理 2 では，処理 1 で選ばれた C_t のサブクラスタ（k 個）の代表ノードと距離測定を行うが，そのうちの 1 つとは処理 1 で距離測定済みであるので，$k-1$ 回の距離測定でよい．処理 3 では N_f との距離測定を行うが，N_f との距離は処理 1 もしくは処理 2 ですでに測定済みである可能性がある．その確率を p とすると距離測定回数は $1-p$ となる．つまり，最上位レベルでの 1 回の近隣度の計算のためには，$k+(k-1)+(1-p)=2k-p$ 回の距離測定が必要である．

次に，すべての最上位クラスタの中で最も近隣度が小さいクラスタが所属クラスタとして選ばれた後で，そのサブクラスタ（C_2）との近隣度を計算する場合を考える．処理 1 ですべてのレベル 3 クラスタの代表ノードと距離を測定するが，その 1 つは C_1 の近隣度計算の処理 1 で測定済みであるため，$k-1$ 回の測定でよい．ただし，C_2 が C_1 の近隣度計算時に C_t として選ばれたものである場合は，すべての代表ノードと距離測定は完了しているため 0 回となる．処理 2，処理 3 に関しては最上位クラスタと同様である．このため，レベル 2 での 1 回の近隣度計算に必要な距離測定回数は，C_1 で C_t として選ばれたクラスタに関しては $(k-1)+(1-p)=k-p$ 回，それ以外のクラスタについては $(k-1)+(k-1)+(1-p)=2k-1-p$ 回となる．レベル 3 以下の中間クラスタでも同様．

末端クラスタでは，処理 1 において，サブクラスタの代表ノードの代わりに全所属ノードとの距離が測定されるという点を除いて，中間クラスタと同様の処理が行われる．処理 2 は省略される．また，処理 1 ですべての所属ノードとの距離は測定済みであるため，処理 3 での距離測定は不要である．

以上の考察を**表 5** にまとめる．これに基づき，あるノードが各レベルで k 個のクラスタと近隣度を計算するために必要な距離測定回数を求めると，最上位レベルでは $k(2k-p)=2k^2-pk$ 回，中間レベルでは（1 つ上のレベルでの近隣度計算時に C_t として選ばれたクラスタが必ず 1 つあるので），$(k-p)+(k-1)(2k-1-p)=2k^2-(p+2)k+1$ 回，同様に末端レベルでは $(k-1)(k-p)=k^2-(p+1)k+p$ 回となる．

この結果から，あるノードが全階層で近隣度を計算するために必要な距離測定回数を求めると，$d(2k^2-(p+2)k+1)-(k^2-3k+2-p)$ となる．

(平成 17 年 7 月 13 日受付)
(平成 18 年 2 月 1 日採録)

上田　達也（学生会員）

平成 17 年大阪市立大学大学院創造都市研究科修士課程修了．同年同研究科博士（後期）課程入学，現在に至る．修士（都市情報学）．ソフトウェア開発やネットワーク管理を中心とした業務をこなしつつ，社会人大学院生として P2P システムの研究に従事．（有）うえだうえおうぇあ取締役社長．

安倍　広多（正会員）

平成 4 年大阪大学基礎工学部情報工学科卒業．平成 6 年同大学大学院博士前期課程修了．同年 NTT 入社．平成 8 年大阪市立大学学術情報総合センター助手．平成 12 年同講師．平成 15 年同大学院創造都市研究科講師．平成 17 年同助教授，現在に至る．博士（工学）．オペレーティングシステム，P2P システム，分散システム管理技術等に興味を持つ．IEEE，電子情報通信学会各会員．

石橋　勇人（正会員）

昭和 62 年京都大学大学院工学研究科修士課程情報工学専攻修了．平成元年同博士後期課程情報工学専攻退学後，京都大学大型計算機センター助手．平成 10 年大阪市立大学学術情報総合センター講師．平成 14 年同助教授．平成 15 年より同大学院創造都市研究科助教授．博士（情報学）．高速ネットワーク，ネットワーク管理システム等に関する研究に従事．人工知能学会，電子情報通信学会，IEEE，ACM 各会員．

松浦　敏雄（正会員）

昭和 50 年大阪大学基礎工学部情報工学科卒業．昭和 54 年同大学大学院基礎工学研究科（情報工学専攻）博士後期課程退学後，同年大阪大学基礎工学部情報工学科助手．平成 4 年同大学情報処理教育センター助教授．平成 7 年大阪市立大学生活科学部教授．平成 8 年同大学学術情報総合センター教授．平成 15 年同大学院創造都市研究科教授，現在に至る．工学博士．ソフトウェア開発環境，ユーザインタフェース，マルチメディア，情報教育等に興味を持つ．ACM，IEEE，電子情報通信学会各会員．