社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－DSM－35 (11)
2004／9／24

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

上田 達也[†,†††] 安倍 広多[†,††] 石橋 勇人[†,††] 松浦 敏雄[†,††]

大阪市立大学大学院創造都市研究科[†] 大阪市立大学学術情報総合センター[††]
(有) うえだうえおうぇあ[†††]

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．本報告では，ノード間のインターネット上の集合を Pure P2P 手法を用いて階層的にクラスタリングするアルゴリズムを提案する．既存の提案手法と異なり，本アルゴリズム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しないため，本手法の適用は容易である．信頼性，スケーラビリティも高い．

A Peer-to-Peer Hierarchical Internet Hosts Clustering Algorithm

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

Graduate School for Creative Cities, Osaka City University[†]
Media Center, Osaka City University[††], Ueda Ueo Ware, Inc.[†††]

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. Since our algorithm does not, unlike any previously proposed one, require any external information of the underlying Internet structure, it is straightforward to apply. Also, reliablity and scalability of our method are high.

1. はじめに

インターネット上のノードを，ネットワーク的に近いもの同士でグループ化する（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性能やスケーラビリティを向上させることができる．クラスタリングは，例えば Web コンテンツなどの分散キャッシュ，複製サーバの効率的な配置，アプリケーションレベルマルチキャスト，Peer-to-Peer(P2P) ファイル共有サービス等の領域で有用である．

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，これまでにも様々な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 方式に基づくものに分けることができる．サーバクライアント方式[1)~4)] では，サーバがクライアント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上で問題がある．

一方，P2P 方式ではサーバを必要としないため，スケーラビリティや信頼性の上で有利である．P2P 方式を用いてクラスタリングを行うものとして，TOPLUS[5)] がある．しかし，TOPLUS を用いて高精度のクラスタリングを行うためには，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るため，実際には TOPLUS の手法を使用することはそれほど容易ではない☆．

本報告では，この問題を解決するアルゴリズムを提案する．提案するアルゴリズムでは，Pure P2P 方式（サーバとなるノードを有しない P2P 方式）により，インターネット上のノード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．BGP ルータから得るような外部情報は不要であり，ノード間の距離さえ測定できればクラスタリング可能である．

☆ TOPLUS はノードの IP アドレスをベースに階層的なクラスタリングを行い，その上でキーの格納・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予め構築しておく．あるいは (2) ノードの IP アドレスを適当な位置 (8, 16, 24 ビット目など) 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抱えている．(1) 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からは必ずしも十分な情報が得られるわけではない．また，BGP ルータでは経路情報の集約 (aggregate) を行うため，幾つかの異なるネットワークが一つの 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り，この方法の適用を困難にしている．また，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，クラスタリングも変化に対応することが望ましいが，この方式はクラスタリング開始前にクラスタ階層を生成してしまうため，動的な変化に対応できない．(2) このように IP アドレスを単純に区切ってクラスタリングを行うと，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（同様のクラスタリングを検討している文献 1) による）．

2. 提案手法の概要

ここでは，提案手法の主な特徴と，採用したデータ構造について述べる．

2.1 特　　徴

提案手法は以下の様な特徴を持つ．

汎用性　BGP ルータから得られる情報の様な外部情報を必要としない．ノード間のネットワーク距離が測定できればクラスタリング可能である．距離の基準としては，ホップ数や RTT(Round Trip Time) 等が考えられるが，どちら側のノードから計測しても同じ値が得られることを前提としている．

スケーラビリティ　実際に使われている P2P システムでは，参加ノード数が数万～数十万に達することがあるが，このように多数のノードを実用的なコストでクラスタリングできる．

対故障性・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．

負荷分散　クラスタリングを行う上で，特定のノードに負荷が集中しない．

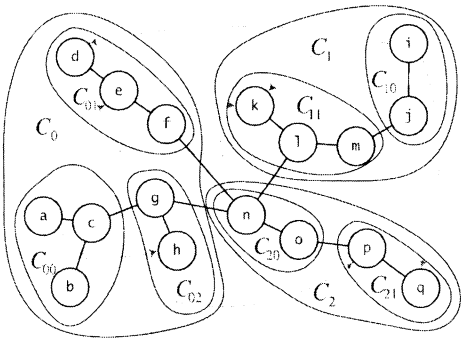
動的クラスタリング　一般的な P2P システムでは，将来参加するノードを予め予想することはできないため，クラスタリングは動的に行う．新たなノードが参加するに従いクラスタは成長し，またクラスタが成長するに従い，より良いクラスタリングのために，ノードは参加するクラスタを切り替える．

階層的クラスタ　P2P によるファイル共有サービスや Web コンテンツの分散キャッシュのようなシステムでは，ノードは自分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．例えば，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便利である．このため，クラスタは階層的とした．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ことになる．また，この構造はスケーラビリティを実現する上でも役立つ．

2.2 クラスタのデータ構造

各ノードで保持するデータ量を削減するため，各ノードは，自分が所属するクラスタの**兄弟クラスタ**（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全て）へのポインタと，所属する末端クラスタの全ノードへのポインタのみを持つ．**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 C_0 に属しているため，その兄弟クラスタである C_1, C_2 のノードへのポインタを持つ (a→k，a→p)．また次のレベル（レベル 2）では C_{00} に属しているため，その兄弟クラスタ C_{01}, C_{02} のノードへのポインタを持つ (a→e，a→h)．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，全ノードへのポインタも持っている (a→c，a→b)．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）．



最上位クラスタを $C_0, C_1, \ldots$
C_i のサブクラスタを $C_{i0}, C_{i1}, \ldots$ と表記している

図 1　クラスタへのポインタの持ち方

あるノードは，所属していないクラスタに属する任意のノードへのポインタを持つ．そのノードを**代表ノード**と呼ぶ．例えば，C_2 に対して，ノード a はノード p を，ノード j はノード q を，それぞれ代表ノードとしている．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．

2.3 クラスタのサブクラスタ数と階層数

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なるべく一定値 (k) となるようにする．

また，(例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 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，用いるデータ構造について 3.1 で述べ，3.2 以降は動作について述べる．

3.1 クラスタ表

個々のノードは，2.2 で述べたクラスタ構造をクラスタ表で管理する（**表 1**）．

表 1 クラスタ表

クラスタの種類	保持する情報
末端以外の所属クラスタ	クラスタ ID，遠方ノードのリスト，代表ノード（常に自分自身）
末端の所属クラスタ	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
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，所属ノード数

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 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 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．

クラスタ表には，他のノードに対して以下の情報を保持する．

- 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）等で使用される．）
- ノードアドレス（IP アドレスなど）
- 最終通信時刻（タイムスタンプ）（3.7 で述べる）
- 自ノードからの距離（代表ノードのみ）
- ノード情報の入手元ノードのアドレス（3.7 で述べる）

3.2 ノード間の通信

ノード間では，**表 2** のメッセージを交換する．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 で述べる）．

表 2 ノード間メッセージ

名前	機能
getTable(宛先クラスタ ID)	クラスタ表を返す
notify(宛先クラスタ ID, 新クラスタ ID, 代表ノードの情報)	新クラスタ生成及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

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一つだけ存在し，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 Pure P2P システムと同様，何らかの手段で既にクラスタリングに参加しているノードへのポインタを少なくとも一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．

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 2 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，そこに新たにクラスタを生成する（3.4.3 参照）．そのクラスタを N が参加するレベル i クラスタとし，探索を終了する．

(4) そうでなければ，3.4.2 で述べる方法で全てのレベル 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，N が所属するレベル i クラスタになる．

(5) ここで，クラスタ表のレベル i のエントリを作成する．この際，自分がもっている代表ノードとバックアップノードの中から 1 つを乱数で選び，それをレベル i クラスタの代表ノードとすることで，負荷分散を図る．

(6) $i = d$ ならば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 は自分が所属する末端クラスタ内の全ノードに表 2 の **notify** メッセージを送ることで，N の参加を通知する．$i < d$ ならば，参加するレベル i クラスタの代表ノードへのポインタを新たに 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．

3.4.2 近隣度の計算

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，(1) クラスタ内で N に最も近いノードとの距離 d_c が近いクラスタを優先する．また，(2)d_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との距離 d_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 $\alpha d_c + d_f$ と定義した（近い方が値は小さくなる）．なお，α は (1) を (2) よりもどの程度優先するかを決めるパラメータである．(1) より (2) 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以下にまとめる．なお，ノード N_1 と N_2 との距離を $\mathrm{dist}(N_1, N_2)$ と表記する．またもっと近いノードと最も遠いノードを正確に求めるにはコストがかかるので，以下で述べる手法では近似的に求めている．

(1) N は，C の代表ノードからクラスタ表を取得し，C の全サブクラスタの代表ノード集合 $\mathcal{S}$ を知る．

(2) N は，$\mathcal{S}$ のそれぞれと距離を測定し，最も近いノード N_c を得る．

(3) N は，N_c からクラスタ表を取得し，N_c の C に対する遠方ノードのリストを得，最も N_c から遠いノードを N_f とする．

(4) 次の式によって N と C の近隣度を計算する．

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

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 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 の全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 2 の **notify** メッセージ）．通知された代表ノードは，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，これと同時に末端クラスタ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てしまう．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 未満に過ぎず，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同士は相手の存在を認識出来ないが，この状況の解消方法は 3.11 で述べる．

3.4.4 遠方ノードの収集

N は，自分が所属しないクラスタ C との近隣度を計算する過程で距離を測定したノードをクラスタ表中の C の遠方ノードリストに登録する．遠方ノードリストは遠い順にソートされて，一定数だけ保持される．

3.4.5 最 適 化

Join では，各レベルで全てのクラスタと 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．

- クラスタを，それぞれの代表ノードとの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 のアルゴリズムで計算し，その値を $E(C_{best})$ 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_{next} 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最小値である．例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，$E(C_{best})$ よりも大きければ，そのクラスタを所属先として選択することはない．このため，そのクラスタとの近隣度の計算は省略する．
- 仮の近隣度が $E(C_{best})$ より小さければ，実際にそのクラスタへの近隣度を計算する．計算値を $E(C_{best})$ と比較し，良いクラスタを C_{best} 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っているため，他ノードへの通知などの処理は行わない．

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，既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．このため，定期的に以下のアルゴリズム (*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）．

Rejoin は，以下に述べる違いを除いて *Join* と同じである．

- 自分が所属するクラスタ C（レベル i）との近隣度を計算する際，N_c を自分自身とすると，近くに C のノードが存在しなくても C への近隣度が小さくなってしまう．このため，クラスタ表上でレベル $i+1$ から d のレベルのクラスタの代表ノード，あるいは末

端クラスタのノードで（自分以外の）最も近いノードを N_c とする．

- 3.4.2 で述べた近隣度計算方法では，クラスタ内で最も近いノードとの距離を近似的に得ているため，本当はもっと近いノードが存在するのに見逃してしまうという事態が起こりうる．この問題を緩和するため，*Rejoin* 時には兄弟クラスタの代表ノードは，候補の中で最も近いものを選ぶ．これにより，*Rejoin* を繰り返すたびにより近いノードを代表として選べるようにする．ただし，*Rejoin* 前と同じ代表ノードが得られた場合は候補から乱数で選びなおす．これは現在の代表ノードの持つクラスタ表からは十分に近いノードが得られない可能性を考慮している．
- 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，(1)*Rejoin* は定期的に行われるので，*Join* 時よりもクラスタ生成の衝突が発生する可能性が高い．(2) あるクラスタ C にノードが 1 つしかない場合に，そのノードが *Rejoin* を行って再度 C に辿り着いたとする．この場合，C にもう一つサブクラスタを生成してそれに引っ越すのは無意味である．

3.7 代表ノードの交代

ノードがクラスタリングから離脱したり，あるいは別のクラスタへ引越しした場合，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喪失してしまう．

このような事態に備えて，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いくつかのノード（**バックアップノード**）の情報を保持しておく．代表ノードを喪失した場合は，バックアップ用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 代表ノード喪失の検出

ノードは，次の契機で自身の代表ノード喪失を検出する．

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 2）．リクエストを受信したノード D は，(引越しなどにより）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には送信ノード N に対してエラーを返す．これによって，ノード N はノード D がすでに目的のクラスタから離脱したことを知ることができる．

また，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

ノードは，代表ノード喪失に備えて，最大 b 個のバックアップノードを保持しておく．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* などの契機で）クラスタ C の代表ノード N_C からクラスタ表を得たとする．このクラスタ表に，N が情報を維持している兄弟クラスタのエントリが含まれている場合，ここからバックアップノード情報を収集する．また，C が末端クラスタでない場合，クラスタ表には C の下位クラスタのエントリが含まれている．これらのエントリに含まれるノードは全て C に属していることが期待できるため，これらもバックアップノードに登録できる．
- N_C が引越しをした後，N が，N_C と通信を試み，宛先クラスタ ID の不一致でエラーとなったとする．このとき，N_C は，エラー情報と共に，N_C 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 の各サブ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，これらも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，既に引越ししている可能性があるので，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渡す際には，タイムスタンプも含めて送信する．バックアップノードリストには最後に通信した時刻の降順で並べておき，一定の数以上の部分は破棄する．

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．例えば *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 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 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 2 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（表 1）．

クラスタリングの全ての処理で，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 の代表ノードを喪失し，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．この方法では，厳密に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 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 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 は見失われる可能性が高くなる．このため，*Join* や *Rejoin* でノード数が b より少ないクラスタに所属したノードは，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広告する．広告を受けたノードは，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, 3.9)．この場合，他に一つでも 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，成功すれば，N のクラスタ表に C のエントリを回復できる．

4. 階層数と最大サブクラスタ数に関する考察

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メッセージ数を最小とするような k と d の値を決定することを考える．

ノードがあるレベルで所属するクラスタを選ぶ際に必要なメッセージ数は約 k^2 であり☆，各階層でクラスタを選択するためには約 k^2d 個のメッセージを要する．ノードが均等に各クラスタに分割される理想的な場合を想定すると，$d = \log_k n$ という関係が成立するため，

$$k^2 \log_k n \qquad (2)$$

を最小化する k の値を求めればよい．式 (2) は，n の値に関わらず，$k = 2$ で最小となる (2 以上の整数の範囲で)．

☆ クラスタ C の近隣度を計算するためには，C の全サブクラスタの代表ノードと距離を測定するが，サブクラスタの代表ノードの 1 つは，C の上位のクラスタの近隣度を計算する段階で既に距離の測定が終わっているため，必要なメッセージ数は $k-1$ である．これに加えて，N_f と距離を測定するためにメッセージが 1 つ必要であり，結局，C を近隣度を計算するためには k 個のメッセージが必要である．1 レベルには k 個のクラスタが存在するのでこの式が得られる．

ノード数の増加に対してメッセージ数の増加は緩やかであり，本手法はスケーラビリティが高いといえる．

5. おわりに

本稿では，Pure P2P 手法により，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．本手法は完全な分散アルゴリズムであり，ノード数に対するスケーラビリティも高い．また，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ため，本手法の適用は容易である．

これから，提案手法のシミュレータを使って，様々な条件のもとで，生成されるクラスタの質や，必要なトラフィック等を評価していく予定である．

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) Krishnamurthy, B. and Wang, J.: On Network-Aware Clustering of Web Clients, *In Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, New York, ACM Press, pp. 97–110 (2000).
2) Bestavros, A. and Mohrotra, S.: DNS-based Internet Client Clustering and Characterization, *In Proceedings of the 4th IEEE Workshop on Workload Characterization (WWC' 01)*, Austin, pp. 159–168 (2001).
3) Agrawal, A. and Casanova, H.: Clustering Hosts in P2P and Global Computing Platforms, *In Proceedings of the Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, Tokyo, pp. 367–373 (2003).
4) Theilmann, W. and Rothermel, K.: Dynamic Distance Maps of the Internet, *In Proceedings of 19th Conf. on Computer Communications (IEEE INFOCOM 2000)*, Vol. 1, Tel Aviv, Israel, IEEE Press, pp. 275–284 (2000).
5) Garcés-Erice, L., Ross, K. W., Biersack, E. W., Felber, P. A. and Urvoy-Keller, G.: Topology-centric look-up service, *In Proceedings of COST264/ACM Fifth International Workshop on Networked Group Communications (NGC)*, Munich, Germany (2003).