peer-to-peerシステム上での評価の高いpeerの発見

山田　太造† 　相原　健郎†† 　高須　淳宏†† 　安達　淳††

† 総合研究大学院大学数物科学研究科　〒 101-8430 東京都千代田区一ツ橋 2-1-2 学術総合センター
†† 国立情報学研究所　〒 101-8430 東京都千代田区一ツ橋 2-1-2 学術総合センター
E-mail: †t_yamada@grad.nii.ac.jp, ††{kenro.aihara,takasu,adachi}@nii.ac.jp

あらまし　P2P システムはここ数年の間で爆発的な成長を遂げている．そこでのユーザ数および共有データ量はともに膨れ上がる一方で，ネットワーク全体の帯域幅消費量も急増している．また，通常の P2P システムではノード間のメッセージ転送で問い合わせを処理するため効率はよくない．そこで本研究では各ノード上に評価の高い他のノード情報が格納された分散インデックス (*Direct Indices*; 以下 *DIs*) を提案し，DIs に示された他のノードに対して直接問い合わせを行う．これにより問い合わせに対する帯域幅消費が減少し，スケーラビリティのあるネットワークの実現が可能となる．また各種実験を行うことでそのパフォーマンスを示す.

キーワード　peer-to-peer，問い合わせ処理，分散インデックス

Efficient Peer Discovery on peer-to-peer Systems

Taizo YAMADA†, Kenro AIHARA††, Atsuhiro TAKASU††, and Jun ADACHI††

† School of Mathematical and Physical Science, The Graduate University for Advanced Studies　National Center of Sciences 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430
†† National Instisute of Informatics　National Center of Sciences 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430
E-mail: †t_yamada@grad.nii.ac.jp, ††{kenro.aihara,takasu,adachi}@nii.ac.jp

Abstract　Recently, P2P systems have been growing explosively and both the number of users and the amount of shared data increase rapidly. It causes the tremendous consumption of bandwidth of the whole network. Query processing of P2P system is inefficient because it propagates query messages in the P2P network. This paper proposes the concept of *Direct Indices(DIs)*, which keep highly evaluated nodes. Since DIs enable peers to access the data directly, bandwith consumption is reduced and scalability of P2P network is achieved. This paper shows the performance of DIs experimentally.

Key words　peer-to-peer systems, query processing, distributed index

1. はじめに

近年，peer-to-peer（P2P）システムは爆発的な成長を遂げている．P2P システムは，役割や能力の等しいノードで構成される分散コンピューティングシステムであり，各ノード間で直接情報のやりとりを行う．そこに参加しているユーザ数，共有データ量は年々増加している．現在存在する P2P システムでは，中央にサーバの振る舞いを行うノード（*peer*; ピア）が存在するシステム（Hybrid P2P;HP2P）と存在しないシステム（Pure P2P;PP2P）の 2 つの種類がある．前者の例として Napster [1] や OpenNap [2] 等が有名である．HP2P システムにおいて各ピアが問い合わせを行う場合，中央に存在するピア（*super peer*）に対して問い合わせを依頼する．super peer はそのネットワーク内に存在する全ピアの情報を保有しており，その情報に基づいて問い合わせ処理を行い，問い合わせを依頼したピアへ結果を返す．後者の例として Gnutella [3] や Freenet [4] 等が有名である．PP2P システムでの問い合わせ処理は，各ピアで問い合わせを解決できない場合，隣接するノードへその問い合わせを転送することによって処理される．

通常の PP2P システムでは，問い合わせは各ピアから隣接する全ピアへ転送されるために，問い合わせのメッセージ数は指数関数的に増加してしまう．このため，ネットワーク全体における帯域幅消費量も莫大な量になっている．P2P システムでは，帯域幅消費を減少させることが重要な課題となっている．各ピアにおいてインデックスを持つ場合では他のピアの情報が分かるため，所望のデータを持っていると思われるピアのみに対して問い合わせを行うことができる．このことにより，ネットワーク上でのメッセージ数は減少する．しかしながら，インデックスを導入した場合，各ピアのインデックスに格納されているピア情報を更新する必要があり，P2P システムでは一般に

更新頻度が高いことが予想される．そのためインデックスを用いる場合はインデックス保有に見合うだけの効果がなければ反対にネットワーク上の帯域幅消費を著しく増加させてしまう危険性がある．

また，現在の PP2P システムでの問い合わせではあまりスケーラビリティがよいとはいえない．それは問い合わせを発行したピア付近でのみ問い合わせ処理を行うためである．そのため問い合わせを発行したピアから比較的離れた場所に所望のデータが存在する場合，その問い合わせを発行した時点ではそのデータの存在を知りうることは不可能に近い．

本研究では P2P システムでの帯域幅消費の軽減と効率的な問い合わせ方法について考察し，各ピア上で効率的な問い合わせ処理を行うための新たな分散インデックス（*Direct Indices*;*DIs*）を提案する．DIs では，各ピアは他のピアの有用度に関する情報を持ち，この情報に基づいて問い合わせを発行/転送するピアを決定する．ピアの有用度はアクセス頻度などに基づいて時間とともに変化し，その情報はピア間で伝播する．有用な情報に対して直接問い合わせを行うことによって，問い合わせの伝播量を減らし，スケーラビリティの改善をもたらす．

しかし，分散インデックスを導入するためには以下の問題点を考慮しなければならない:

- 各ピアでのインデックスの更新時（$join/leave/update$）の帯域幅消費を抑えながら，効率的でスケーラビリティのあるインデックス構築可能なメカニズムを導入する必要がある．
- 問い合わせ発行時の帯域幅消費を減少させるメカニズムを導入する必要がある．インデックスを用いない方法に比べネットワーク全体の帯域幅消費を減少させるため，更新と問い合わせのトレードオフの関係を考慮する必要がある．

本論文では，まず 2 節では関連研究をサーベイする．次に 3 節では DIs の概念を導入し，そのアルゴリズムを示す．4 節では実験のための各設定を示す．5 節では DIs を用いた場合とそうでない場合の比較実験を行ない，DIs を用いることでネットワークパフォーマンスが改善されることを示す．また対象となる P2P システムは PP2P システムとする．

2. 関 連 研 究

P2P システムは，インデックスの観点から以下のように分類できる．まず第 1 のグループは，インデックスを *super peer* で集中管理する HP2P システムのタイプで，Napster [1] や OpenNap [2] などがあげられる．しかし，このタイプのシステムは，*super peer* が停止してしまうとシステム全体も停止してしまう．そのため，*super peer* の管理コストの問題がある．

第 2 のグループとして，インデックスを導入していない PP2P システムで，Gnutella [3] はこの種類に属する．このタイプのシステムは問い合わせの伝播が全方位に行われるため，問い合わせのメッセージ数が膨大になり，システム全体のパフォーマンス低下を招く欠点がある．

第 3 のグループは，各ピアがインデックスを保持する PP2P システムで，Freenet [4] などが例としてあげられる．Freenet では各ピアは最近参照されたデータをキャッシュに蓄える．しかし，このインデックス内に存在しないデータを検索する場合の検索方法は考慮されていない．この問題に対して Yang 等は，routing indices（RIs）を提案している [5]．このインデックスでは，P2P システムが扱うデータのカテゴリを定義し，このカテゴリごとにデータを多く有するピアの情報をインデックスとして用いる．しかし，あらかじめ各データを分類する方法を考慮する必要がある．また，スケーラビリティに優れているとは言い難い．

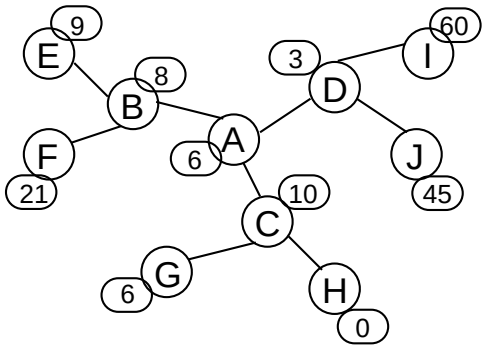

図 1　Direct Indices の例（$fanout = 3$）

この他にも P2P システム上での効率の良い情報検索法の研究が行われており，[6] ではインデックスを用いた方法と用いない方法を示している．また [7]〜[12] は P2P システムでの効率的なデータ検索法を提案している．これらの研究のアプローチは我々が行った方法とは異なったアプローチで．特別なネットワーク構造を用い，そのネットワーク構造内で効率的なデータ検索の方法を提唱している．しかしこれらの研究では，その特定のネットワーク構造内でのみ有効な方法である．

また，P2P システムでのインデックスの問題は分散データベースの問題とも関連する [13]．しかしながら分散データベースでの各ノードはシステムへ長時間参加状態にあり，スタティックである．またシステムに参加しているノード数は P2P システムに比べかなり少数である．このことにより，分散データベースに用いられるインデックスの方法はそのまま P2P システムへ適応するのは必ずしも適切ではない．

3. Direct Indices（DIs）

本節では，*Direct Indices*（*DIs*）の概念とそのアルゴリズムを提案する．

3.1 Direct Indices の概念と 3 つのセッション

P2P システムの分散インデックスでは，通常各ピアは半径 r 以内にあるピアの情報をインデックスとして保持する．ここでのピアの情報としては，そのピアが持つデータのメタデータ（データ名，データサイズ，作成時間等）や，そのピア自身の情報（ping の結果，IP アドレス等）がある．例えば，*local indices* [6] を用いた場合，以下の問い合わせ処理が行なわれる．ピア p が問い合わせを行う場合，まず p 自身のインデックスを使って問い合わせ処理を行う．ここで該当するデータがあればそのデータを結果として返す．また，返された結果が不十分な場合は，一定の規則に従って他のピア p' に問い合わせを転送す

表 1　ピア A の *Direct Index*

UID	value	location
A	6	local
I	60	D
J	45	D
F	21	B
B	18	neighbor
C	12	neighbor
G	6	C

る．p' はそのインデックスに対し問い合わせ処理を行う．これを繰り返す．転送規則の例として，問い合わせを行うピアと転送を打ち切るピアをホップ数で指定するものがある．例えば，問い合わせを行うピアは，問い合わせを発したピアから 1 および 5 ホップで到達できるピアであり，5 ホップを越えて問い合わせを転送しないといった規則が用いられる．P2P システムでは隣接ピアのネットワークへの参加・離脱などのタイミングでインデックスが更新されその更新が伝播する．この更新メッセージはピアの半径 r が大きければそれだけ増加することになる．

本論文では，各ピアは P2P ネットワーク上で下記の処理を通してインデックスやデータをやりとりするものとする．

- P2P システムに参加する *join session*
- 問い合わせを行う *query session*
- データを取得する *transport session*

各ピア p は隣接するピア p' から有用なピアの情報をピアの集合 $rec(p', p)$ として受け取ることによってインデックスを取得・更新する．インデックス情報の受取りは join session および query session で行われる．有用なピアの情報は各ノードごとに有用度に基づいてランキングされて格納されており，ピア p が問い合わせを行う場合は，そのうちの上位 n 個のピアに対して問い合わせを行う．問い合わせは，結果に満足するまで繰り返す．このとき，問い合わせを行ったピアから結果とともにピアの有用度の情報を検索結果と合わせて取得しインデックスを更新する．インデックスが更新されるとその情報が隣接ノードに伝播する．

例として図 1 のネットワークを考える．この図において各丸に囲まれたものをピア，そこに付随してある数値をそのピアの有用度，各ピア間の線分はそのピア間で join の関係にあることを示す．このときピア A のインデックスは表 1 のように表される．このインデックスを *Direct Index*（*DI*）と呼ぶことにする．DI はピア情報を格納する．このピア情報はピアのネットワーク上の識別名，有用度，位置を 1 レコードとして格納される．ここでの位置はこの DI を保有するピアからどの方向に存在するかという情報である．この値は *local*（DI 保有のピア），*neighbor*（隣接ピア），*UID*（隣接ピアの方向に存在）のいずれかとなる．また，有用度の決定については後述する．

3.2　ピアの有用度

各ピアを評価するためにピアに有用度を設定する．本研究において，各ピアの有用度はそのピア保有のデータの有用度及び保有データ数によって評価されるものとした．各データの有用度は各データの需要によって決定する．各データの需要はそのデータの参照回数によって判断する．しかしながら，各データの需要は時間の経過とともに変化する．そのため参照回数が比較的多いデータであっても，現在はあまり需要がない可能性がある．よって，データ公開からの時間経過とともにそのデータの有用度を下げる．これにより，新規性は高いがあまり参照されていないデータの有用度を高く設定することが可能となる．また，各ピアの保有データ数も考慮する必要があると考えられる．

表 2　ピア A の有用度

UID	value
B	9.333
C	8.667
D	12

以上を踏まえ，ピアの有用度は以下のように定義される．まず，各ピアは，以下の式で定義されるそのピア固有の有用度 $E_{local}(p)$ を持つ．

$$E_{local}(p) \equiv \sum_{d \in D(p)} \frac{d_{ref} + 1}{|D(p)| \cdot \log(t_c - d_m + e)} \quad (1)$$

ここで，$D(p)$ は p が保有するデータの集合であり，d_{ref} はデータ d の参照回数，d_m は d を最後に更新した時刻である．また，t_c は現在の時刻である．

次に，各ピアからその隣接ピアへ渡される有用度について考える．本研究ではここでの有用度を先ほどの $E_{local}(p)$ をそのまま用いなかった．その理由は以下の通りである．ネットワーク上の有用度が比較的高い特定のピアに対して問い合わせメッセージの集中が起こる危険性がある．その危険性を回避するためにメッセージを程良く分散することが必要となる．またネットワーク上の各ピアにおいて，そのピアの隣接ピアを考慮する必要性がある．あるピア p の各隣接ピアは比較的有用であれば，p はそのネットワーク上での要所であると判断されるためである．そのため各ピアの隣接ピアを考慮した有用度を設定する．

以上を考慮し，ピア p_1 からピア p_2 へ渡される p_1 の有用度 $E_{global}(p_1, p_2)$ を以下のように定めた．

$$E_{global}(p_1, p_2) = E_{local}(p_1) + \sum_{p \in Nei(p_1)} \frac{E_{global}(p, p_1)}{F} \quad (2)$$

ここで，$Nei(p)$ は p_1 の join session で隣接するピアで p_2 以外のピア p の集合を表している．また，F は各ピアの隣接ピア数の最大値である．

たとえば，図 1 においてピア B，C，D におけるピア A の有用度は表 2 のように表される．この例からも分かるように，ピアの有用度はそのピアの情報を格納するピアの位置によって変化する．また，この技法は PP2P システムにおいてのみ可能なピアの評価方法であり，HP2P システムでは困難である．それは，HP2P システムでは各ピアの情報は *super peer* のみによって維持されるため，各 peer の評価はそのシステム内において一意に決定されてしまうためである．

3.3　join session

join session は主にピア間の更新メッセージの伝播に用いられ

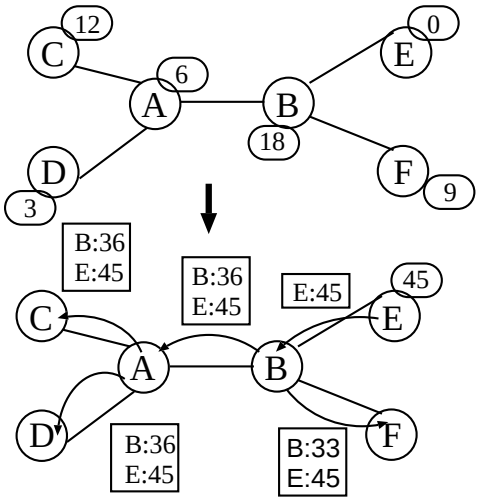

図 2　update の例（$fanout = 3$）

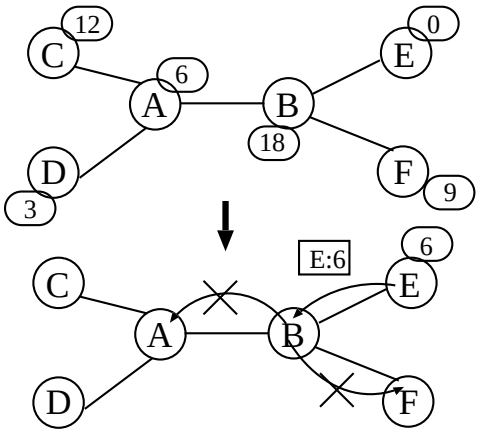

図 3　update の例（$fanout = 3$）

る．更新は以下のように行われる．まず，更新が必要な状況になると，ピア p は $E_{global}(p, p_i)$ を計算する．ここで p_i は p の隣接ピアである．この値と DI に格納されている各ピアの有用度の内で上位 m 個を p が推奨するピア集合 $rec(p, p_i)$ とする．$rec(p, p_i)$ と p の P2P システム上での識別子を一組としたものを更新メッセージ $update(p_{uid}, rec(p, p_i))$ とする．このメッセージを受け取ったピア p_i は p_i の DI を更新する．さらに p_i は更新が必要になれば同様に更新の手続きを行う．

更新が必要かどうかの判定はその更新状況による．たとえば，データが追加されたが推奨するピアには影響がない場合は更新メッセージを送信しない．

たとえば，図 2 においてピア E に更新が起こったとする．すると，この更新により E は更新メッセージは B へ送信される．この例では各矩形内の情報が更新メッセージに相当する．そして，B でも更新が起こるため，B の隣接ピアへ更新メッセージを送信する．これを繰り返す．次の更新の例として図 3 においてもピア E にて更新が発生したとする．すると E の隣接ピアである B へ更新メッセージを送信される．しかしながら，B では推奨するピアに影響はなかったためこれ以上の更新メッセージの伝播は行われない．

次に，ピア間の連結（*join*）について述べる．join は更新の一

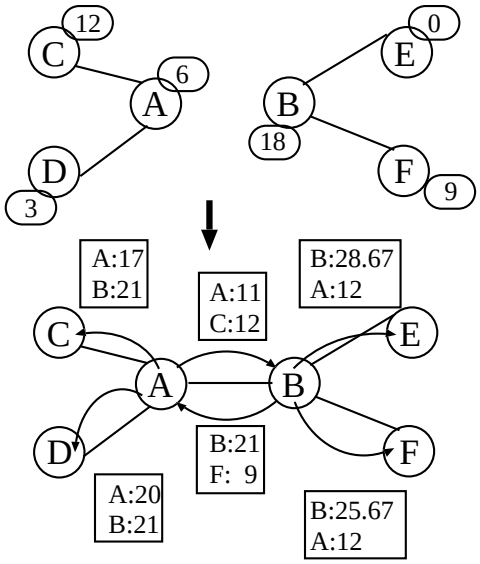

図 4　join の例（$fanout = 3$）

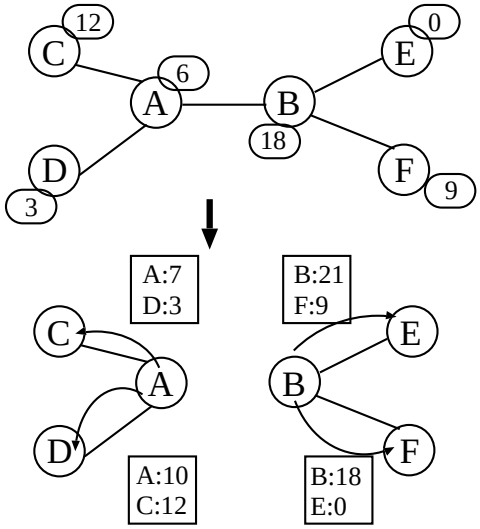

図 5　leave の例（$fanout = 3$）

種であると考える．たとえば，p_1 と p_2 の $join$ ではそれぞれ相手ピアへの更新メッセージを作成し，互いにその更新メッセージの交換に成功すれば，join に成功したとする．そして，p_1 と p_2 は各 DI を更新し，必要があれば更新メッセージをそれぞれの join session の隣接ピアへ送信する．

たとえば，図 4 においてピア A とピア B の間での join を行うとする．このとき A，B ともそれぞれ相手ピアへの更新メッセージを作成し，そのメッセージを交換する．このメッセージの交換に成功すれば join に成功したこととなる．そして，この例ではこれ以上の更新が発生するため A は C と D へ，B は E と F へ更新メッセージを送信する．

ピア間の切断（*leave*）は join と逆の動作である．leave も join と同様に更新の一種であると考える．それぞれのピア間で leave を行う場合，それぞれのピアの DI から leave する方向にあるピア情報を削除すれば十分である．

図 5 においてピア A とピア B の間で leave が発生した場合を例に取る．まず，A と B の DI 内のそれぞれ相手の情報及び

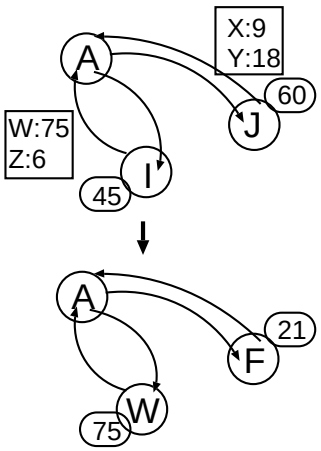

図 6　query の例

その相手方向にあるピアの情報も併せて削除する．更新が必要となれば更新を行う．A は C と D へ，B は E と F へそれぞれ新たな更新メッセージを送信する．これで leave は完了する．

3.4　query session

問い合わせは以下のように処理される．問い合わせが発行されると，まずピア内にあるデータに対しての問い合わせ処理を行う．この問い合わせで十分な検索結果が得られれば処理を終了する．不十分な場合は問い合わせの転送を行う．このとき問い合わせを発行するピアの DI に格納されたピアで有用度が上位 n 以内にあるピアに対して問い合わせが行われる．問い合わせを受けたピアは，問い合わせ処理を行い，その結果とともにそのピアの推奨ピア集合も添付して送信する．問い合わせ結果を受け取ったピアは十分なデータが得られれば処理を終了する．不十分な場合は，問い合わせ結果とともに受け取った推奨ピア集合と DIs に存在するピア集合との和集合から問い合わせ送信済みのピア集合を除いたピア集合に対して有用度の上位 n 個のピアに問い合わせを発行する．この操作を十分な結果が得られるまで続ける．

たとえば，図 1 におけるピア A が問い合わせを行う場合を考える．これを図 6 に示す．まず A のローカルにて解決しようとする．解決すれば，問い合わせは終了となる．解決しなければ他のピアへ問い合わせを転送する．問い合わせの転送先の候補は A の DI でのランキングの上位 n 個内にあるピアとなる．$n = 2$ であれば，まず 2 つのピアへ問い合わせを行う．この例では転送先は I，J となる．I と J に問い合わせメッセージを送信し，問い合わせの結果を得る．このとき，問い合わせ結果とともに，I と J からそれぞれのピアが推奨するピアの情報を A は得ることになる．そして，A は結果に対して満足すれば問い合わせは終了する．満足しなければ，結果とともに得たピア情報と A の DI に格納されていてまだ問い合わせ転送を行っていないピアから転送先のピアを選出する．この場合，I，J からそれぞれピア X と Y，W と Z のピア情報を得た．これと A の DI に格納されているまだ送信していないピア集合から F と W が選出される．再び問い合わせの転送を行い，以上を繰り返すことで問い合わせ処理を行う．

表 3　実験に用いる各定数

定数	値
ピア数	10000
共有データの種類	10
共有データのオリジナル数	300
結果の満足数	20
$join\ session$ での推奨ピア数	2
$query\ session$ での推奨ピア数	2

表 4　実験に用いる各変数

変数	説明
$init\_contents(p)$	p の初期共有データ
F_{join}	ファンアウト数（join session）
F_{query}	ファンアウト数（query session）
$number_{query}$	発行する問い合わせ数

4. 実　　験

4.1　実験の概要

本研究の実験では各問い合わせ方法において以下の点について注目し計測を行った．

- 問い合わせ転送時の帯域幅消費量
- 問い合わせを満足するまでの時間
- DIs を用いた問い合わせの更新時の帯域幅消費量

問い合わせ転送時の帯域幅消費量を計測するに当たり，問い合わせの転送回数を計測することで代用した．それは，各転送のメッセージデータの大きさはあまり差がないためである．そのため転送のメッセージ数を計測することで帯域幅消費計測と同等の計測を実現することが可能である．

次に問い合わせを満足するまでの時間を計測であるが，一般に P2P システムにおいて問い合わせ処理の処理時間は，その大部分がネットワーク上での通信時間である．またネットワークの接続環境（ダイアルアップ，xDSL，ケーブル，ISDN，FTTH 等）の差異により問い合わせ処理時間は左右する [5]．このことより，問い合わせを満足するまでの時間を計測するに当たり，計算機上の計算時間やネットワーク上での通信時間による処理時間を計測のではなく，問い合わせの発行回数を計測することとした．各問い合わせ方法において，問い合わせを発行し返された結果に対し満足するかどうかを調べ，満足すればそのときの問い合わせの発行回数を計測した．ただしインデックスを用いない方法では上記のことが行えないため，一定間隔で問い合わせを満足したかどうかを判定し，満足すれば返答したピアのホップ数の最大値を計測することで代用した．

最後の項目はインデックスを用いた場合のみの問題である．前述したようにインデックスを用いた場合，その更新のために帯域幅消費が著しければインデックスを用いない方が帯域幅消費が軽減する場合がある．そのためインデックスを用いた問い合わせ処理では重要な項目であると考えられる．この計測も問い合わせ転送の帯域幅消費計測と同様に更新のメッセージ数を計測することで代用する．

表 3 と表 4 はそれぞれ実験に用いた各定数及び各変数を表し

表 5　計測項目

変数	説明
$time_{forward}$	問い合わせ転送回数
$time_{query}$	問い合わせ発行回数
$number_{update}$	更新メッセージ数
$number_{forward}$	問い合わせの転送回数

ている．また，表 5 は実験の計測項目を表している．ここでは，各ピア p の各データの参照回数は 0 から 500 までの値，公開時間は 0 から 6.048×10^8 までの値とした．

また各実験において，ある問い合わせ q が満足されるとは，q の結果数が結果の満足数以上返されたことを意味する．

実験を行うに当たり，P2P システムでの問い合わせ方法として次の 3 つを用意した．

- *Direct Indices* を用いる
- *Iterative Deepening* を用いる
- index を用いない

最初の方法は本研究の分散インデックスを用いた問い合わせ法である．最後の方法は gnutella [3] に用いられているものである．この方法では問い合わせの転送先は全隣接ピアであり，転送された問い合わせはそのピアで処理するとともに転送も行う．2 番目の方法も問い合わせ先は全隣接ピアとなるが，それ以上の転送は行わずその問い合わせを凍結させる．問い合わせの発行ピアが結果に満足すればこの処理は終了する．満足しなければ，同様の問い合わせを再度転送する．再度問い合わせのメッセージを受け取ったピアは，問い合わせの処理は行わず，先ほど凍結した問い合わせを継続し，単に隣接ピアへ問い合わせを転送する．以上を繰り返すことで問い合わせを満足しようとする方法である. [6]

また，システム全体の初期のデータ分布を決定する必要がある．これを以下のように決定する．

- *uniform*
- *80/20*

前者は，システム上の全データが偏りなく散らばって存在する場合である．この場合，各ピア p の $init\_contents(p)$ の数は 0 から 100 までの値とする．後者は，システム上のデータ分布に偏在が生じる場合である．システム上の全ピアの 20%にシステム全体のデータの 80%が存在するように各ピア p の $init\_contents(p)$ を設定した．

また，各ピアにて問い合わせを行う際に，各問い合わせ方法において問い合わせ処理を完了した後に，得られた結果のうち 1 から 10 程度のデータを得るように設定した．

4.2　問い合わせの転送

まず，F_{join} を 3 から 8 として各問い合わせ法について問い合わせの転送回数を計測した結果がそれぞれ図 7 と図 8 である．このとき $F_{query} = F_{join}$ である．

この値は 1 ピアあたりの問い合わせ転送メッセージ数を示している．インデックスを用いない場合，一旦問い合わせを発行すればその問い合わせを凍結することができないため，問い合わせの転送回数は莫大な量になった．*Iterative Deepening* を用いた場合，問い合わせが満足すれば問い合わせを凍結することが可能であるため問い合わせ転送回数は減少した．DIs を用いた場合は問い合わせ対象のピアを限定するため，*Iterative Deepening* を用いた場合よりも問い合わせメッセージ数を減少させることが可能になった．

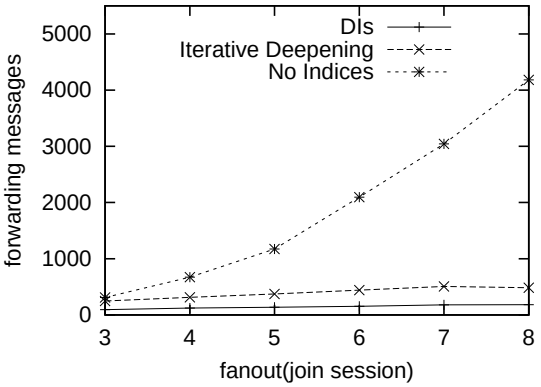

図 7　平均転送回数（*uniform*）

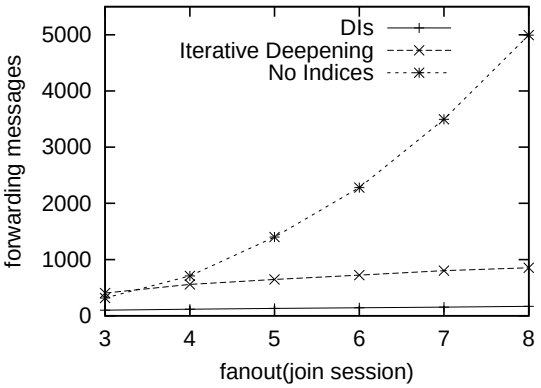

図 8　平均転送回数（*80/20*）

データ分布による差異は以下の通りである．*Iterative Deepening* を用いた問い合わせでは，データ分布に偏在があるときは偏在のない分布の場合に比べ転送メッセージ数が多くなった．これは問い合わせを発行するピアの周囲に所望のデータを有するピア数が少ない場合があり，そのため問い合わせメッセージを比較的多く伝播した結果であると考えられる．図 8 において $F_{join} = 3$ ではインデックスを用いない場合よりも転送メッセージ数が多くなった．これは，*Iterative Deepening* はピアの凍結と解凍のため問い合わせメッセージを繰り返し送信する必要があることが原因であると思われる．DIs を用いた問い合わせはデータ分布にあまり影響を受けなかった．これは，DIs に格納されたピア情報は比較的離れた位置に存在したピアの情報をも格納可能であったということを示唆している．また，このことより DIs をもちいたシステムはスケーラビリティのあるネットワークであるといえる．他の問い合わせ方法を用いた場合，データ分布に偏在があるとき，すなわち問い合わせを発行したピア周辺には所望のデータを有するピアが存在しないとき，問い合わせのための転送回数が増える．しかしながら DIs を用いた場合では，問い合わせを行ったピアの周辺に所望のデータを有するピアが存在しない場合でも，あまり問い合わせの転送回

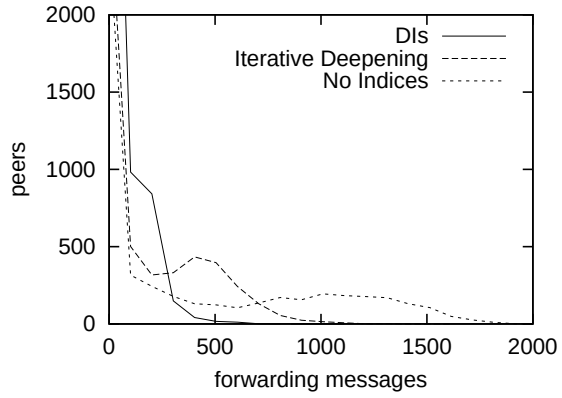

図 9　転送回数分布（*uniform*）

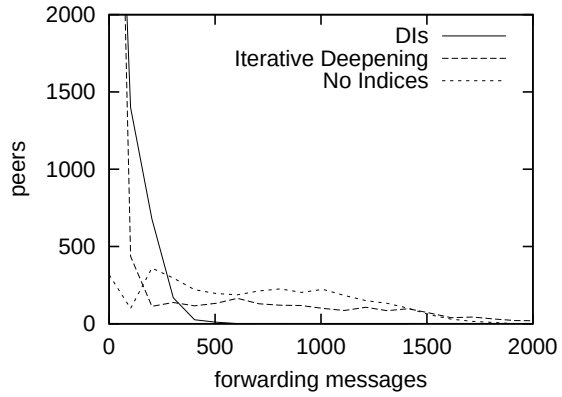

図 10　転送回数分布（*80/20*）

数は増加しなかった．このことより，DIs を用いることでシステムのスケーラビリティは向上することを示すことができる．

次に各問い合わせ方法での各ピアの問い合わせの集中度合いを示す．これを図 9 と図 10 にて示す．DIs を用いた問い合わせの場合，データ分布に偏在がある場合は偏在のない場合に比べ，問い合わせの集中度合いが多少高いが，各データ分布とも比較的問い合わせの集中が起こったピア数は少なかったといえる．これは $E_{global}(p_1, p_2)$ にて，各ピアの有用度は他のピアそれぞれにて異なった値で格納されるためである．他方，*Iterative Deepening* を用いた場合とインデックスを用いない場合では，問い合わせの集中のが起こったピアが多く存在した．これは，問い合わせメッセージが多くのピアへ伝播されたためである．特に，データ分布に偏在がある場合において，*Iterative Deepning* を用いた問い合わせはインデックスを用いない方法の結果にかなり近づく結果を示した．この場合，*Iterative Deepning* を用いた問い合わせはあまり効率的ではない手法であることが分かる．それは，問い合わせを発行したピアから比較的近くに所望のデータを有するピアが存在しない場合，多くの問い合わせメッセージを発行する結果を招くためである．

4.3　インデックスの更新

次にインデックス更新のメッセージ数を示す．F_{join} を上記と同様に変化させ，DIs を用いて場合の分散インデックス更新のメッセージ数を計測した．その結果が図 11 である．ここで示してある結果は 1 ピアあたりの join と leave 以外の更新メッセージ数である．図 7 と図 8 の DIs の値に図 11 の値を加えてもネッ

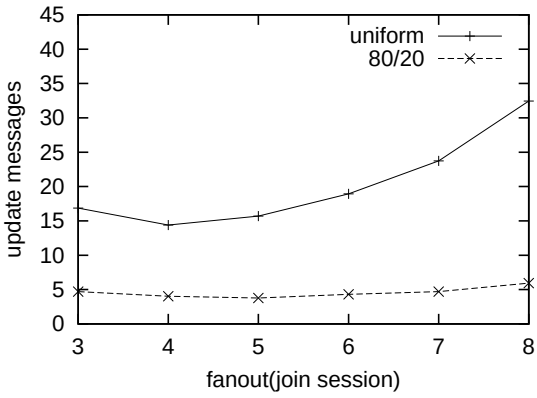

図 11　平均更新メッセージ数（$number_{query} = F$）

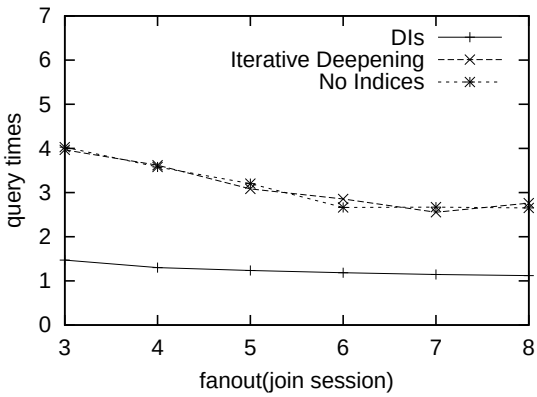

図 12　平均発行回数（*uniform*）

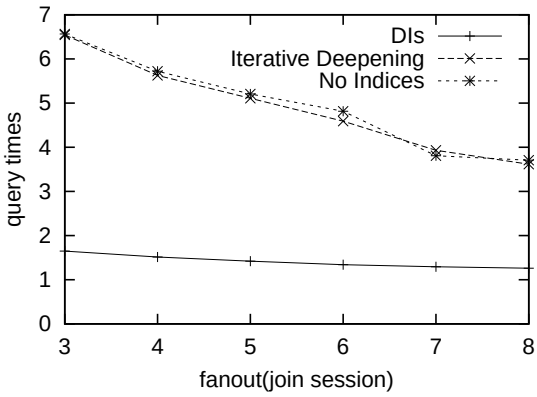

図 13　平均発行回数（*80/20*）

トワーク全体のメッセージ数は減少する結果となった．DIs を更新したピアが更新メッセージを送信する可能性がある場合，まず送信する前に送信先のピアに対する前回の送信情報を比較する．比較の結果，あまり差異がない場合は更新メッセージを送信しない．このことより，不必要と思われる更新メッセージの送信を未然に防ぐことができている．

またデータ分布に偏在がある場合は特に更新メッセージが少ない．この場合，有用度の高いピアの情報が DIs に格納された場合，その値は絶対的なものになるため更新メッセージがあまり発生しないことを示す．

4.4　問い合わせの発行

各問い合わせ法において各データ分布にて問い合わせ結果が

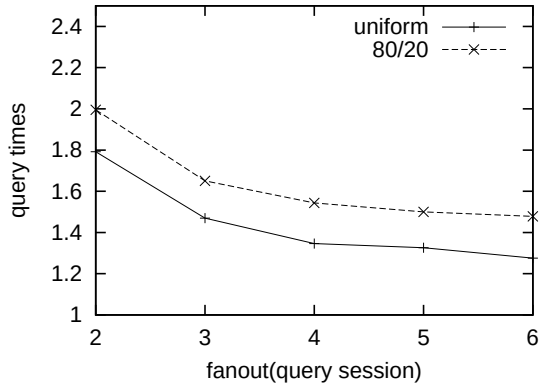

図 14　平均発行回数（$F_{join} = 3$）

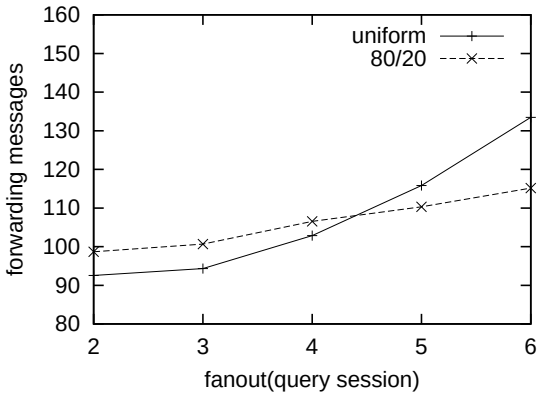

図 15　平均転送回数（$F_{join} = 3$）

満足されるまで問い合わせを繰り返した結果を図 12 と図 13 に示す．ここでは，DIs は他の政策よりも問い合わせを繰り返す回数が圧倒的に少ない．これは，DIs に格納されいる情報が問い合わせ先の選択に非常に有効であることを示している．

また，DIs を用いた場合ではデータ分布の差異の影響をあまり受けず，F_{join} 及び F_{query} を変化させたとしてもあまり変化はなかった．*Iterative Deepening* を用いた場合及びインデックスを用いた場合では，データ分布の影響を受ける．これらの理由は前述の通りである．また，F_{join} の値が大きければそれだけ問い合わせの発行回数は少なくすむ結果となった．この理由は自明であろう．

図 14 は DIs を用いた問い合わせを行った場合において，F_{query} の値を変化させたときの問い合わせ発行回数である．各データ分布においても同様の傾向を伺うことができる．図 15 は F_{query} を 2 から 6 まで変化させたときの問い合わせ平均転送回数を示す．F_{query} は問い合わせメッセージを転送する対象ピア数である．よって，この値が高いほど転送メッセージ数は多くなる．

この 2 つの図はちょうどトレードオフの関係となっている．つまり，問い合わせの発行対象のピアを増加させることで満足するだけの結果を得るまでの時間は短縮することはできるが，問い合わせによる帯域幅消費量は増加してしまう．この場合（$F_{join} = 3$）では $F_{query} = 3$ が適切な値であろうと推測している．F_{join} の値を 3 から 8 まで変化させたところ，$F_{join} = F_{query}$ が適切な値であった（図は示していない）．

5. おわりに

現在，P2P システムは一般的に認知されはじめ，Napster や Gnutella を代表とするデータ共有システムをはじめグループウェア [14] やオンラインコミュニケーションツール [15] 等，多様な形式で用いられはじめている．しかしながら，P2P システムでは効率的な問い合わせと帯域幅消費の軽減の 2 つの問題がある．そこで本研究ではこの問題を解決するために分散インデックスである *Direct Indices*（*DIs*）を導入した．また，DIs を用いた場合とインデックスを用いない場合での各種実験を行った．その結果，DIs を用いた場合ではインデックスを用いない場合に比べ，問い合わせの転送メッセージ数を軽減し，規定以上の問い合わせ結果を返す時間も改善された．また，実際に DIs を利用するにあたり課題とされる項目として，特定のピアに対する問い合わせ集中の発生という問題があった．しかしながら実験の結果，DIs は問い合わせを分散させることが可能であるということが分かった．これにより，P2P システム全体のパフォーマンス改善されることを確認した．

文　献

[1] Napster, “http://www.napster.com/”.

[2] OpenNap: Open Source Napster Server, “http://opennap.sourceforge.net/”.

[3] Gnutella website, “http://www.gnutella.com/”.

[4] FreenetProject website, “http://freenet.sourceforge.net/”.

[5] Arturo Crespo, Hector Gracia-Molina, “Routing Indices For peer-to-peer Systems”, Proc. The 22nd International Conference on Distributed Computing Systems (ICDCS), Vienna, AUSTRIA, July 2002.

[6] Beverly Yang, Hector Gracia-Molina, “Improving Search in peer-to-peer Networks”, Proc. The 22nd International Conference on Distributed Computing Systems (ICDCS), Vienna, AUSTRIA, July, 2002.

[7] S.Ratnasamy, P.Francis, M.Handley, R.Larp, S.Shenker, “A scalable content-addressible network”, In ACM SIGCOMM, San Diego, USA, August 2001.

[8] J.Kubiatowicz, D.Bindel, Y.Chen, S.Czerwinski, P.Eaton, D.Geels, R.Gummadi, S.Rhea, H.Weatherspoon, W.Weimer, C.Wells, B.Zhao, ”OceanStore: An Architecture for Global-Scale Persistent Storage”, Proc. the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Cambridge, USA, November 2000.

[9] B.Zhao, J.Kubiatowicz, A.Joseph, “Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing”, U. C. Berkeley Technical Report UCB//CSD-01-1141, April 2000.

[10] Y.Chen, R.Katz, John D. “Dynamic Replica Placement for Scalable Content Delivery”, Proc. 1nd Internal Workshop on Peer-to-Peer systems(IPTPS), Cambridge, USA, March 2002.

[11] I.Stoica, R.Rorris, D.Karger, M.F.Kaashoek, H.Balakrishnan, “Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications”, In ACM SIGCOMM, San Diego, USA, August 2001.

[12] A.Rowstron, P.Druschel, “Pastry: Scalable, dissributed object location and Routing for large-scale peer-to-peer systems”, In Middleware, Heidelberg, Germany, November 2001.

[13] D.kossmann, “The State of the Art in Distributed Query Processing”, In ACM Computing Surveys, vol.32, no.4, pp.422-469, December 2000.

[14] Groove Networks, Inc., Desktop Collaboration Software, “http://www.groove.net/”.

[15] ICQ Inc., “http://web.icq.com/”.