Ringed Bloom Filterによる分散ハッシュテーブルのトラフィック量削減

清 雄一[†] 松崎和賢[†] 本位田真一[†,††]

Peer-to-Peer コンテンツ共有システムを実現する一手法として，分散ハッシュテーブル（DHT）がある．DHT を用いることで，存在するコンテンツを確実に発見することができるが，コンテンツの全文検索を行うにはスケーラビリティに欠ける．複数語による AND 検索を行う際，コンテンツ数が多くなるとコンテンツ ID を送信するためのトラフィック量が大量に発生してしまうという問題があるからである．既存研究では，Bloom Filter という集合要素の圧縮手法を用いて，AND 検索時にコンテンツ ID を送信するための大量のトラフィック量を削減している．だが Bloom Filter には欠点があり，トラフィック量の削減が十分ではない．本研究では，新たに Ringed Bloom Filter を提案し，さらにトラフィック量を削減できることを示す．

Using Ringed Bloom Filters to Reduce the Communication Traffic on a Distributed Hash Table

Yuichi Sei,[†] Kazutaka Matsuzaki[†] and Shinichi Honiden[†,††]

A distributed hash table (DHT) technology realizes peer-to-peer contents sharing systems. A DHT system can find all files if the files are registered, but it lacks scalability at full text searching. In multi-word searching, there is so much communication traffic for transmission of file IDs. In related works, the authors reduced the amount of the communication traffic by using a bloom filter which is an ingenious randomized data-structure for concisely representing a set in order to support approximate membership queries. However, bloom filters have a limited role if several sets have different numbers of elements. Accordingly, we propose a "ringed bloom filter" to solve the problem of normal bloom filters.

1. はじめに

Peer-to-Peer システムは，中央サーバがなくてもコンテンツやサービスの共有が可能な分散ネットワークである．Peer-to-Peer システムの一形態として，分散ハッシュテーブル（DHT）[10),12),16)] がある．DHT を用いることにより，検索対象のコンテンツを保有するノードを，速く正確に発見することが可能となる．

DHT では，主にメタデータや，コンテンツに含まれている単語による検索を行い，コンテンツを発見する．現状では，動画等のマルチメディアコンテンツであれば，タイトル等ごく少量のメタデータに対してのみ検索が行える．だが将来的には，タイトル以外にも，検索の効率化のために多数のメタデータが付加されるようになり，これらを対象に検索が行えるようになると考えられる[2)]．本研究においては，テキストを含むコンテンツのキーワード検索を対象としている．つまり，テキストドキュメントであればそれに含まれている全単語を対象とし，動画コンテンツであれば，それに付加されている単語のメタデータすべてが対象である．

DHT では，コンテンツとノードのそれぞれにハッシュ値を割り当て（コンテンツに割り当てられるハッシュ値を，**コンテンツ ID** と呼ぶ），コンテンツのハッシュ値に最も近いハッシュ値を持つノードにそのコンテンツを格納する．単語からコンテンツを検索するためには，単語それぞれにハッシュ値を割り当て，単語の数だけコンテンツをノードに登録することになる．検索する際には，検索したい単語のハッシュ値を計算し，近いハッシュ値を持つノードに問い合わせることで検索を行う．ある単語 w のハッシュ値を，$h(w)$ と表すことにする．$h(w)$ 担当ノードには，w を含むすべてのコンテンツ ID が登録される．

DHT では指定した複数の単語を持つコンテンツを検索する AND 検索を行うときに，大量のトラフィック

† 東京大学
The University of Tokyo
†† 国立情報学研究所
National Institute of Informatics

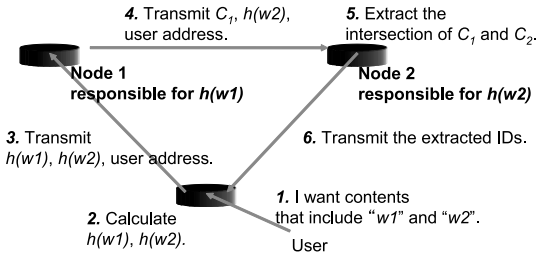


図 1 単純な AND 検索手法である Naive Algorithm の手順（ユーザは $w1$ と $w2$ の 2 つの単語を含むコンテンツを検索）
Fig. 1 The process of naive algorithm: normal searching for multi-word text (here, user wants files that contain the two words "$w1$" and "$w2$") on a DHT.

量が発生するという問題がある．DHT における AND 検索時の処理の流れを**図 1** に示す．単語 $w1$ を含むコンテンツのコンテンツ ID の集合を C_1，単語 $w2$ を含むコンテンツのコンテンツ ID の集合を C_2 と表現している．$h(w1)$ 担当ノードである Node 1 は C_1 を，同様に Node 2 は C_2 をそれぞれ保持しており，目標は，$C_1 \cap C_2$ の抽出である．この手法を，**Naive Algorithm** と呼ぶ．Naive Algorithm では，Node 1 から Node 2 に C_1 を送信するときに多大な無駄なトラフィック量が発生する．なぜなら一般に，$w1$ を含むコンテンツ数（$|C_1|$）は，$w1$ と $w2$ を同時に含むコンテンツ数（$|C_1 \cap C_2|$）よりもはるかに多いからである．文献 7) によると，2003 年当時の Google [1] にインデックスされているページを DHT で実現すると，1 回の AND 検索のクエリにつき，平均 530 MB のトラフィック量が発生する．コンテンツを検索するだけで大量のトラフィック量が発生し，多大な検索時間がかかってしまうことになる．本研究では，このように，各ノードに膨大な数のコンテンツが登録されている状況を想定する．

この課題に取り組む既存研究の多くは，Bloom Filter [3),4)] という大規模な集合を表現するデータ構造を用いている．Bloom Filter で表現された集合に対しては，ある要素が含まれているかどうかの判定のみを行うことができる．このとき，ある一定の誤差率（False Positive [☆] Rate，FPR）を許容することで，Bloom Filter のビット数を大幅に削減することができる．Bloom Filter は，その使い方から 2 種類に分類される（Variable-size Bloom Filter と Fixed-size Bloom Filter．次章参照）．だが，Variable-size Bloom Filter は計算時間が膨大にかかるので，コンテンツ検索には利用することができない．一方，Fixed-size Bloom Filter の FPR は Variable-size Bloom Filter よりも大きい．そこで，本研究では Variable-size Bloom Filter なみの FPR を実現し，かつ Fixed-size Bloom Filter と同等の時間で計算可能な，Ringed Bloom Filter を提案する．Ringed Bloom Filter を用いることで，Bloom Filter を用いた既存研究よりもトラフィック量を削減できることを示す．

本論文の構成を以下に記す．2 章において，DHT における AND 検索時のトラフィック量削減に取り組む既存研究や，既存研究で使われている Bloom Filter について述べる．3 章において，Ringed Bloom Filter の提案を行う．4 章において，Ringed Bloom Filter の性能評価を行う．5 章では，Ringed Bloom Filter を DHT に適用した実験，評価を行う．6 章で考察を記し，7 章で本論文のまとめを記す．

2. 関 連 研 究

2.1 AND 検索時のトラフィック量削減

文献 6) では，単語の集合そのものを登録し，ノード間に特別な関係を持たせることで効率的な検索を実現している．この手法は，ユーザが多くの語による AND 検索を実行する際には有効だが，1 語〜3 語程度の少数の語による検索では，有効に機能しないと筆者自身も指摘している．文献 11) によると，Web の検索エンジンに投げられた 99,405 のクエリを調査した結果，ユーザによる検索の 78%は 3 語以下の検索である．

pSearch [18)] ではコンテンツのセマンティクスに従って，分散ハッシュテーブルのインデックス付けを行っている．また，文献 17) では，コンテンツの要約を分散ハッシュテーブルの key としている．これらの手法は，類似検索には有効であるが，本研究が対象としている，「あるキーワードを含むコンテンツ検索」のような要求を満たすことができない．

DHT における AND 検索時のトラフィック量削減に取り組んでいるその他多くの既存研究 [7),11),13),19)] では，Bloom Filter というデータ構造を用いている．

文献 7) や 11)，19) では，後述する Transmission Filter Algorithm（TFA）やそれを併用した手法を提案している．これらの研究においては，Fixed-size Bloom Filter が用いられている（**TFA with Fixed-size Bloom Filter**）．

本研究の目標は，提案する Ringed Bloom Filter を TFA へ適用することで（**TFA with Ringed Bloom Filter**），TFA with Fixed-size Bloom Filter よりも

☆ 本来，集合には含まれていないのに，含まれていると誤判断すること．

トラフィック量の削減を行うことである．したがって TFA with Fixed-size Bloom Filter を採用している既存研究のそれぞれにおいて，Fixed-size Bloom Filter を，Ringed Bloom Filter へ置き換えることで，よりトラフィック量を削減することができる．

文献 13) では，独自の Filter を TFA に適用している．だがパラメータの設定が困難であるという課題がある．文献 14) によると，共有されるコンテンツの特徴（コンテンツの数や，コンテンツに含まれているキーワード数）があらかじめ明確である場合はトラフィック量の削減が可能だが，コンテンツの特徴が不明である場合は，Fixed-size Bloom Filter を用いた手法と同等量しか削減できないと指摘している．コンテンツの特徴があらかじめ明確であるという制約は厳しく，本研究では，コンテンツの特徴があらかじめ分からないような状況を想定する．

TFA with Fixed-size Bloom Filter の詳細を以下に述べる．図 1 において，Node 1 から Node 2 へコンテンツ ID の集合 C_1 を送る代わりに，C_1 から作成した Fixed-size Bloom Filter を送信する．Fixed-size Bloom Filter を受け取った Node 2 は，「$C_1 \cap C_2$ および False Positive によって誤検出された集合」を抽出することができる（この 2 つの集合を区別することはできない）．抽出したコンテンツ ID の集合を Node 1 へ送信することで，Node 1 は $C_1 \cap C_2$ であるコンテンツ ID を正確に抽出できる．

本論文で提案する，TFA with Ringed Bloom Filter では，Fixed-size Bloom Filter の代わりに Ringed Bloom Filter を用いる．

2.2 Bloom Filter

2.2.1 Bloom Filter の構造

Bloom Filter の作成手順は次のようになる．まず，すべての値が 0 にセットされている m ビットのベクトル v を用意する．また，k 個の独立したハッシュ関数，$h_1, h_2, \ldots, h_k$ を用意する．各ハッシュ関数は 0, $1, \ldots, m-1$ の値を返すものとする．集合 $A = \{a_1, a_2, \ldots, a_n\}$ の Bloom Filter を作成する際は，各要素 $a \in A$ に対して $h_1(a), h_2(a), \ldots, h_k(a)$ を計算し，v 内の $h_1(a), h_2(a), \ldots, h_k(a)$ 番目のビットをそれぞれ 1 にセットする（このとき，あるビットは複数回 1 にセットされることになる）．集合 A の全要素に対してこの計算を行うことで，集合 A の Bloom Filter が生成される．

ある要素 b が集合 A の要素であるかどうかのテストを，集合 A の Bloom Filter に対して行う手順は次のようになる．まず，要素 b の k 個のハッシュ値 $h_1(b), h_2(b), \ldots, h_k(b)$ を計算する．そして，集合 A の Bloom Filter の該当部分がすべて 1 であるかどうかをチェックする．1 つでも 0 が存在すれば，要素 b は確実に集合 A に含まれていないと判断できる．もし，該当部分がすべて 1 であれば，要素 b は集合 A に含まれていると推測できる．だが，ある確率のもとでこの推測は間違いである．この誤判断を行う確率を，False Positive Rate（FPR）と呼ぶ．

FPR は k，m，n の関数であり，

$$FPR \approx (1 - e^{-kn/m})^k \tag{1}$$

と表される[4]．式 (1) は，明らかに $k = \ln 2 \times m/n$ のとき最小値をとる．このとき，

$$FPR = (1/2)^k \tag{2}$$

となる．目標 FPR を α とすると，k が整数値をとることを考慮して，

$$k = \lfloor \log_{1/2} \alpha \rfloor \tag{3}$$

となる．また，これらより，次式が導かれる．

$$m = M(\alpha, n) = \lfloor \lfloor \log_{1/2} \alpha \rfloor \times n / \ln 2 \rfloor \tag{4}$$

式 (4) のように，Bloom Filter のビット長 m は，集合の要素数 n に依存する．このように，集合の要素数に従って，m を変化させて作成する Bloom Filter を特に，Variable-size Bloom Filter と呼ぶ．一方，集合の要素数にかかわらず，m を一定として作成する Bloom Filter を，Fixed-size Bloom Filter と呼ぶ．Fixed-size Bloom Filter を作成する際には，あらかじめ共通の Filter ビット長を決定しておく必要がある．

2.2.2 Bloom Filter の課題

Fixed-size Bloom Filter の課題

Fixed-size Bloom Filter は Variable-size Bloom Filter よりも FPR が大きいという欠点がある[9]．異なる大きさの集合 N 個から，それぞれ Fixed-size Bloom Filter を作成することを考える．大きい集合から作成された Fixed-size Bloom Filter の FPR は大きく，小さい集合から作成された Fixed-size Bloom Filter の FPR は小さい．また式 (1) より，FPR の値が小さいときは，集合の要素数が増えるに従って，FPR が指数関数的に増大することが分かる．したがって集合の要素数にばらつきがある場合，平均すると Fixed-size Bloom Filter の FPR は増大する傾向にあることが分かる（4 章参照）．

Variable-size Bloom Filter の課題

一方，Variable-size Bloom Filter は Fixed-size Bloom Filter よりも多大な計算時間がかかるという欠点がある[15]．Bloom Filter の作成時や，ある要素が

Bloom Filter の構成要素に含まれているかどうかの判定時には，該当要素のハッシュ値を計算する必要がある．ハッシュ値の計算は時間がかかるので，DHT で Bloom Filter を利用する際には，あらかじめ各要素のハッシュ値を計算しておく必要がある．Bloom Filter を送る側のノードでは，あらかじめ Bloom Filter を作成することができる．だが，この Bloom Filter のビット長が決まっていないと，Bloom Filter を受け取る側のノードで，各要素のハッシュ値を計算することができない．なぜなら，2.2.1 項で説明したように，要素のハッシュ値は Bloom Filter のビット長 m に依存するからである．したがって，Variable-size Bloom Filter を用いる場合には，各要素のハッシュ値をあらかじめ計算しておくことができず，多大な検索時間を要することになる（4 章参照）．

3. Ringed Bloom Filter の提案

Fixed-size Bloom Filter と Variable-size Bloom Filter の持つ欠点を克服するために，Ringed Bloom Filter を提案する．式 (4) より，ある α を満たすために必要な Filter のビット数 m は集合の要素数 n とほぼ比例関係にある．したがって，

$$M(\alpha, n) = n \times M(\alpha, 1) \quad (5)$$

の関係が成り立つ．このように，n 個の要素を持つ集合 A に対して，要素 1 つずつから $M(\alpha, 1)$ ビットの Fixed-size Bloom Filter を n 個作成することができる．

ある要素 b が，この集合の構成要素かどうかを検査するとき，n 個のうち 1 つの Filter に対しての FPR は α となる．だが要素 b が集合 A の要素であるかどうかを検査するためには，n 個すべての Filter に対して検査を行う必要がある．この場合 FPR は $\alpha' = (1 - (1 - \alpha)^n)$ となる．α が十分小さい値のときは，$\alpha' \approx n \times \alpha$ となり，FPR が約 n 倍になる．これを避けるためには，ある要素 b に対して検査を行う Filter をただ 1 つに限定する必要がある．

情報を付加せずに Filter を 1 つに限定するため，n 個の要素をそれぞれのハッシュ値☆に基づいて n グループに分類し，それぞれのグループから固定長の Filter を作成するようにする．これを n-divided bloom filters（nDBFs）と呼ぶことにする．各要素は，そのハッシュ値に基づいてグループ分けされるので，各グループに 1 つずつ要素が振り分けられるとは限らないことに注意していただきたい．nDBFs の具体的な作成手順は次のようになる．すべての値が 0 にセットされている γ ビットの n 個のベクトル $v_0, v_1, \ldots, v_{n-1}$ を用意する．ここで，目標とする FPR を α とすると

$$\gamma = M(\alpha, 1) \quad (6)$$

に設定する．また $k + 1$ 個の独立したハッシュ関数 $h_0, h_1, \ldots, h_k$ を用意する．h_0 は，$0, 1, \ldots D-1$ (D は十分大きな数)，その他のハッシュ関数は，$0, 1, \ldots, \gamma - 1$ の値を返すものとする．集合 $A = \{a_1, \ldots, a_n\}$ の nDBFs を作成する場合は，各要素 $a \in A$ に対し $h_0(a)$ の計算を行う．次に $T = h_0(a) \bmod n$ を計算し，ベクトル v_T に対して，通常の Bloom Filter 作成の操作を行う．集合 A の全要素に対してこの処理を行うことで，集合 A の nDBFs が生成される．ある要素 b がこの nDBFs の構成要素であるかどうかのチェックを行う手順は次のとおりである．まず，b のハッシュ値 $h_0(b)$ を計算し，n 個の Filter のうち b が含まれうる Filter をただ 1 つに限定する．その後，通常の Bloom Filter に対する判定を行う．

1 つの nDBFs は n 個の固定ビット長 ($= \gamma$) の Filter から作成されている．したがって，この Filter を受け取る側は，あらかじめ各要素のハッシュ値を計算しておくことができるので，Fixed-size Bloom Filter なみの計算速度が期待できる．また，仮に n 個の各 Filter の構成要素が 1 つずつであるならば，全体として Variable-size Bloom Filter と同等の FPR が実現できる．だが実際にはこのようにはならない．$h_0(a)$ に従って各要素を振り分ける際，均等には振り分けられないからである（**図 2**）．図では，FPR が大きい Filter を濃い黒色で表現している．図内の Case 1 が望ましい状態であるが，n 個の要素をハッシュ値に基づいて n 個のグループに分けると，実際には Case 2

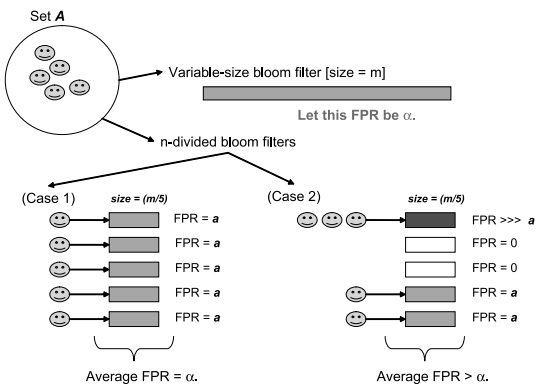


図 2 n-divided bloom filters（nDBFs）の作成とその課題
Fig. 2 Procedure and problem of n-divided bloom filters (nDBFs).

☆ ここで用いるハッシュ関数は，DHT システムで用いるハッシュ関数や，Bloom Filter を作成するときに用いる k 個のハッシュ関数とは独立したものを用いる．

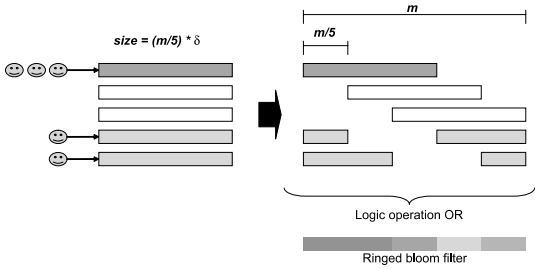


図 3 Ringed Bloom Filter の作成手順
Fig. 3 Procedure of a ringed bloom filter.

のように各グループに含まれる要素数にはばらつきが生じる．このとき，いくつかの Filter は FPR が大きく，いくつかの Filter は FPR が小さい．平均すると FPR は増大する傾向にある．望ましい状態は，n 個の Filter すべての FPR が等しい状態である．この状態はつまり，値が 1 であるビットの出現割合が，どの Filter においても等しい状態である．

この状態を実現するために，nDBFs における各 Filter を重ね合わせるという処理を行う（**図 3**）．1 つ 1 つの Filter において 1 であるビットの出現割合に偏りがあっても，複数の Filter を重ね合わせることでその影響を緩和することが目的である．概念的には，n 個の各 Filter のビット長を $\delta \times \gamma$ に設定して nDBFs を作成し，γ ビットずつずらして重ね合わせる．また最終的に作成される Filter の全長を

$$m = n \times \gamma = n \times M(\alpha, 1) = M(\alpha, n) \quad (7)$$

ビットに設定し，各 Filter を重ね合わせる際には，円形をなすようにする．したがって，この Filter 長は，Variable-size Bloom Filter の Filter 長と等しい．これを **Ringed Bloom Filter** と呼ぶ．上記の集合 A から Ringed Bloom Filter を作成する具体的な手順は次のようになる．すべての値が 0 である $n \times \gamma$ ビットのベクトル V と，$k+1$ 個の独立したハッシュ関数 $h_0, h_1, \ldots, h_k$ を用意する．h_0 は，$0, 1, \ldots D-1$（D は十分大きな数），その他のハッシュ関数は，$0, 1, \ldots, \delta \times \gamma - 1$ の値を返すものとする．各要素 $a \in A$ に対して $h_0(a)$ の計算を行い，$T_a = h_0(a) \bmod n$ を計算する．各ハッシュ関数 $h \in \{h_1, \ldots, h_k\}$ に対し，

$$\tau = \gamma \times T_a + h(a) \bmod (\gamma \times n) \quad (8)$$

を計算し，ベクトル V 内の τ 番目のビットをそれぞれ 1 にする．集合 A の全要素に対してこの処理を行うことで，集合 A の Ringed Bloom Filter が生成される．ある要素がこの Ringed Bloom Filter の構成要素であるかどうかのチェックも同様に行う．

4. Ringed Bloom Filter の評価

本研究で提案する Ringed Bloom Filter と，従来技術である Fixed-size Bloom Filter，Variable-size Bloom Filter について比較を行う．比較項目は，FPR と計算時間である．FPR の比較を行うために，まず Ringed Bloom Filter の FPR の計算式を導出する．

4.1 Ringed Bloom Filter の False Positive Rate

ハッシュ関数の数 k，要素数 n，パラメータ δ のもとでの，Ringed Bloom Filter の FPR を求める．FPR の観点では，要素 n 個をハッシュ値に基づいてグループ分けすることは，ランダムにグループ分けすることと同等である．

Ringed Bloom Filter において，$[\gamma \times i \bmod |V|,\ \gamma \times (\delta + i) - 1 \bmod |V|]$（$i = 0, 1, \ldots, n-1$）に相当するビットを抽出したものを，$\Theta_i$ とおく．Θ_i のビット長は，$\gamma \times \delta$ である．さらに Θ_i を γ ビットずつ δ 個の部分に分割し，それぞれを $\Gamma_{i,j}$ とおく（$j = 0, 1, \ldots, \delta - 1$）．各 $\Gamma_{i,j}$ は，δ 個の Filter が重なり合って導出されたものである．この δ 個の Filter がそれぞれ，$\zeta_{i,j,p}$ 個の要素から作成されているとすると，$\Gamma_{i,j}$ は，$N[\Gamma_{i,j}] = \sum_{p=0}^{\delta-1} \zeta_{i,j,p}$ 個の要素から作成されていることになる．また，

$$\bar{N}[\Gamma_i] = \sum_{i=0}^{\delta-1} N[\Gamma_{i,j}]/\delta \quad (9)$$

とおく．このとき，Θ_i における FPR は，式 (1) より，

$$\Re_{\Theta_i}(k, \gamma) = (1 - e^{-k \times \bar{N}[\Gamma_i]/\delta \times \gamma})^k \quad (10)$$

となる．

$\bar{N}[\Gamma_i]$ の値が Q となる確率を $P(\bar{N}[\Gamma_i] = Q)$ とすると，Ringed Bloom Filter の FPR は，

$$\Re_{RBF} = \sum_{Q=\min Q}^{\max Q} \left[P(\bar{N}[\Gamma_i] = Q) \times \Re_{\Theta_i}(k, \gamma) \right] \quad (11)$$

となる．ここで，$\min Q$ は Q がとりうる最小値を，$\max Q$ は Q がとりうる最大値を表す．

次に，$P(\bar{N}[\Gamma_i] = Q)$ を求める．これは $P(N[\Gamma_{i,j}] = Q)$ を算出することによって求まる．

(1) n 個の箱 $Box_0, Box_1, \ldots, Box_{n-1}$ を円形に並べ，その中の 1 つの箱 Box_0 に注目する．

(2) n 個の箱のうち，連続した δ 個の箱をランダムに選択し，それぞれに要素を 1 個ずつ入れる．

- $n \leq \delta$ の状況を考慮すると，すべての箱に

少なくとも $\lfloor \delta/n \rfloor$ 個の要素が入る．

- したがって，Box_0 に $\lfloor \delta/n \rfloor + 1$ 個の要素が入る確率は，$(\delta \bmod n)/n$ であり，$\lfloor \delta/n \rfloor$ 個の要素が入る確率は，$(n-(\delta \bmod n))/n$ である．

(3) (2) を n 回繰り返す．

(4) Box_0 に $t + \lfloor \delta/n \rfloor \times n$ 個の要素が入っている確率は，$N[\Gamma_{i,j}] = t + \lfloor \delta/n \rfloor \times n$ である確率に等しく，

$$P(N[\Gamma_{i,j}] = t + \lfloor \delta/n \rfloor \times n) = {}_nC_t \times (\frac{\delta \bmod n}{n})^t \times (\frac{n-(\delta \bmod n)}{n})^{n-t} \quad (12)$$

である．また，t のとりうる範囲は $0 \leq t \leq n$ である．

式 (10)，(11)，(12) より，Ringed Bloom Filter の FPR は，

$$\Re_{RBF}(k,n,\delta,\gamma) = \sum_{t=0}^{n} \Big[\frac{{}_nC_t \times (\delta \bmod n)^t \times (n-(\delta \bmod n))^{n-t}}{n^n} \times (1-e^{-k\times(t+\lfloor \delta/n \rfloor \times n)/(\delta\times\gamma)})^k \Big] \quad (13)$$

となる．

4.2 Ringed Bloom Filter のパラメータの決定

Ringed Bloom Filter を作成する際は，まず目標とする FPR を設定する必要がある．これを α とする．γ は，α と式 (6) より決定される．

次に，パラメータ k を決定する．式 (13) において，δ を n より十分大きな値に設定する状況を考える．$t \leq n \ll \delta$ であるから，$-k\times(t+\lfloor \delta/n \rfloor \times n)/(\delta\times\gamma) \rightarrow -k/\gamma$ となる．したがって，

$$\Re_{RBF}(k,n,\delta,\gamma) = \Big[\sum_{t=0}^{n} \frac{{}_nC_t \times (\delta \bmod n)^t \times (n-(\delta \bmod n))^{n-t}}{n^n} \Big] \times (1-e^{-k/\gamma})^k \quad (14)$$

二項定理より，$\sum_{t=0}^{n} {}_nC_t \times (\delta \bmod n)^t \times (n-(\delta \bmod n))^{n-t} = n^n$ であるので，

$$\Re_{RBF}(k,n,\delta,\gamma) = (1-e^{-k/\gamma})^k \quad (15)$$

となる．したがって，$k = \ln 2 \times \gamma$ のとき，FPR は最小値をとる．さらに，式 (4)，(6) と，k が整数値であることを考慮すると，

$$k = \lfloor \log_{1/2} \alpha \rfloor \quad (16)$$

となる．したがって，δ を n より十分大きな値に設定している状況では，k を式 (16) に設定することで，Ringed Bloom Filter の FPR を最小化することができる．

次に，パラメータ δ の特性を調べる．**図 4** は，各要素数のもとで作成した Ringed Bloom Filter の FPR を式 (13) より計算したものである．α は $1/2^{10}$ に設定した．γ は式 (6) から，k は式 (16) からそれぞれ計算した．また，$\delta = 1$，10，100，1,000 に設定した．図から，δ を大きい値に設定するほど FPR が減少することが読み取れる．したがって，δ は十分大きな値に設定すべきであることが分かる．本論文では，以降 $\delta = 100{,}000$ に設定した．

このように，目標 FPR を α と設定すると，調節可能なパラメータ γ，k，δ は，集合の要素数にかかわらずに決定される．また，式 (13) におけるパラメータ n は与えられた集合の要素数であり，調節することはできない．

したがって，Ringed Bloom Filter を作成する際に定めなければならないパラメータは，α のみである．また，Ringed Bloom Filter の全長 m は，式 (7) より求められる．

4.3 Ringed Bloom Filter と従来の Bloom Filter との比較

4.3.1 False Positive Rate の比較

図 5 は，$\alpha = 1/2^{10}$ のもとでの各 Filter の FPR を表している．Fixed-size Bloom Filter と Variable-size

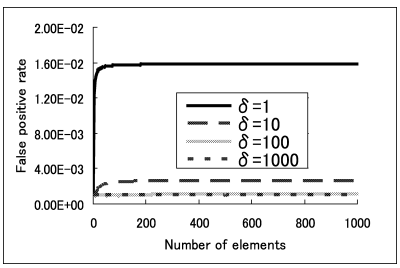


図 4 δ の値が，Ringed Bloom Filter の FPR に与える影響
Fig. 4 The impact of δ on the FPR of a ringed bloom filter.

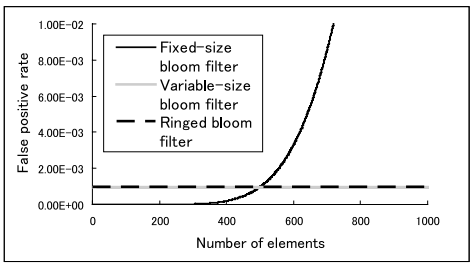


図 5 各 Filter の FPR
Fig. 5 A FPR of the each filter.

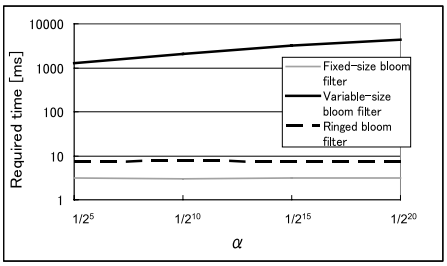


図 6 ある要素が，与えられた Filter の構成要素かどうかのチェックにかかった時間（要素数は 10^6 個）
Fig. 6 Required Time for checking whether elements are members of a filter (number of elements is 10^6).

Bloom Filter の FPR は，式 (1) から求め，Ringed Bloom Filter の FPR は式 (13) から求めた．Ringed Bloom Filter は，$\delta = 100{,}000$ に設定した．また合計 Filter ビット長を等しくするため，Fixed-size Bloom Filter については式 (1) において $n = 500$ に設定してある．各 Filter の FPR の平均値は Variable-size Bloom Filter を 1 とすると，Fixed-size Bloom Filter は約 9.37，Ringed Bloom Filter は約 1.0000004 であった．これらから，Fixed-size Bloom Filter の FPR は Variable-size Bloom Filter と比較して大きいことが分かる．また，Ringed Bloom Filter の FPR はほぼ Variable-size Bloom Filter に等しいことが分かる．

4.3.2 計算時間の比較

図 6 は，1 つの Filter に対して，10^6 個の要素がその Filter の構成要素に含まれているかどうかの判定にかかった時間を表している．α の値を変えながら実験を行った．また，Fixed-size Bloom Filter と Ringed Bloom Filter に関しては，10^6 個の要素のハッシュ値をあらかじめ計算しておいた．Variable-size Bloom Filter に関しては，各要素のハッシュ値の計算時間も結果に含めている．図より，Fixed-size Bloom Filter に比べて Variable-size Bloom Filter の計算には多大な時間を要することが分かる．一方，α にかかわらず Ringed Bloom Filter は Fixed-size Bloom Filter と近い時間で計算が可能であるといえる．

5. DHT における Ringed Bloom Filter の評価

実験は，トラフィック量削減の手法を用いない Naive Algorithm，2 章で説明した Transmission Filter Algorithm（TFA）について行った．TFA に関しては，従来技術である Fixed-size Bloom Filter を用いたもの（TFA with Fixed-size Bloom Filter）と，本研究の提案である Ringed Bloom Filter を用いたもの（TFA with Ringed Bloom Filter）でそれぞれ実験を行った．前章で示したように，Variable-size Bloom Filter に関しては計算時間の問題があるので使用しない．この 3 つの手法について，トラフィック量の比較を行った．

表 1 使用した論文の特性
Table 1 The characteristics of the published papers.

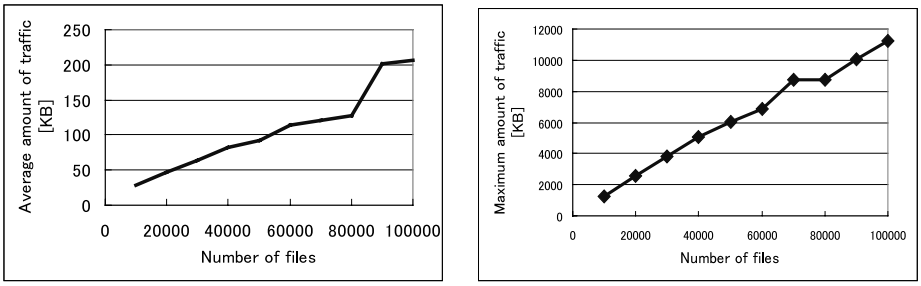
Number of files	100,000
Number of words, irrespective of how many times it occurs in all contents	47,086
Average number of words per content	632.7
Average number of registered contents per word	1343.8

5.1 実験設定

実験に用いるコンテンツとして，英語論文を 10 万本用意した．論文は，コンピュータ科学・医学・経済学等の分野から，1980 年から 2005 年の間に書かれたものを集めた．各コンテンツから，語彙データベース WordNet [8] に含まれる名詞・動詞・形容詞に限定して抽出を行い，それらをキーとしてノードに登録した．**表 1** にコンテンツの性質を示す．各ユーザは，用意したコンテンツのうちランダムに単語を 2 語選択し，つねに AND 検索することとした．DHT ハッシュ関数として，一般によく用いられるハッシュ関数である SHA-1 [5] を使用した．この SHA-1 は 160 ビットの値を返すので，コンテンツ ID は 160 ビットの容量を持つ．トラフィック量の計算に関しては以下のように行う．仮に，ユーザが $w1$ と $w2$ の 2 語による AND 検索を行ったとする．Naive Algorithm については，$h(w1)$ 担当ノードから $h(w2)$ 担当ノードへ送信するコンテンツ ID の容量とする．TFA については，$h(w1)$ 担当ノードから $h(w2)$ 担当のノードへ送信する Filter のビット長を $T1$ とし，$h(w2)$ 担当ノードから $h(w1)$ 担当ノードへ送り返すコンテンツ ID の容量を $T2$ とするとき，AND 検索に必要なトラフィック量を $T1 + T2$ と定義した☆．

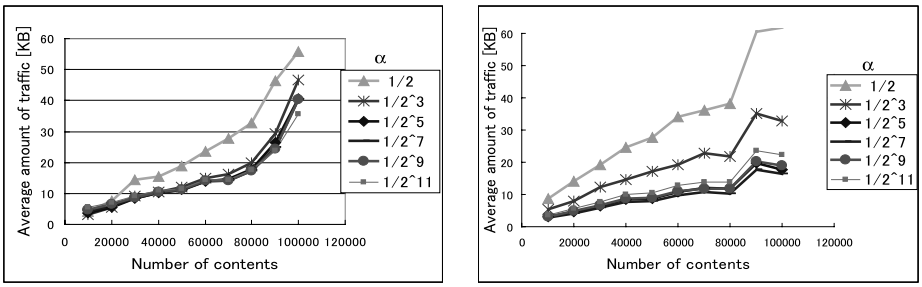
Fixed-size Bloom Filter を用いる場合には，Filter ビット長を全ノードで共通に設定する必要がある．あらかじめ実験を行い，このビット長を，コンテンツ数が 50,000 のときにトラフィック量が最も削減できる長さになるように設定した．このとき，$m = 2{,}164$ であっ

☆ Fixed-size Bloom Filter では，集合の要素数（ここでは，ノードに保存されているコンテンツ ID の数）にかかわらず $T1$ は固定であるが，$T2$ が変動することになる．また，Ringed Bloom Filter では，式 (7) より，集合の要素数に応じて $T1$ も変動する．



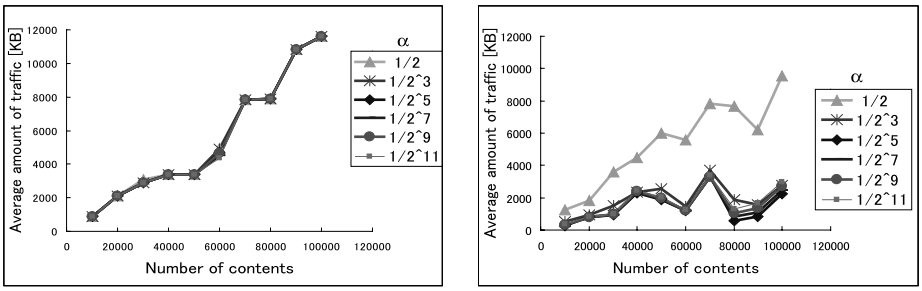
(a) Result of the average amount of traffic (b) Result of the maximum amount of traffic

図 7 Naive Algorightm におけるトラフィック量
Fig. 7 Amount of traffic using a naive algorithm.



(a) Result of TFA with Fixed-size Bloom Filter (b) Result of TFA with Ringed Bloom Filter

図 8 各 Filter を利用した際の平均トラフィック量
Fig. 8 Average amount of traffic.



(a) Result of TFA with Fixed-size Bloom Filter (b) Result of TFA with Ringed Bloom Filter

図 9 各 Filter を利用した際の最大トラフィック量
Fig. 9 Maximum amount of traffic.

た．また，α をさまざまに変化させて実験を行った．

5.2 実験結果

コンテンツ数を 10,000 から 100,000 まで変化させてトラフィック量を測定した．各コンテンツ数において，5,000 回の試行を行った．Naive Algorithm における結果を，**図 7** に記す．図 7 より，コンテンツ数が増加するほど，トラフィック量の平均値，最大値ともに増加することが分かる．また，コンテンツ数が 100,000 であるときの，1 回のクエリにおける平均トラフィック量は約 200 KB，最大トラフィック量は約 11,000 KB であった．

TFA について，α の値を 1/2 から $1/2^{11}$ まで変えながら測定した結果を，**図 8** と**図 9** に示す．

図 8 (b) より，Ringed Bloom Filter を用いる手法では，コンテンツ数にかかわらず，α が $1/2^7$ であるときに平均して最もトラフィック量を削減できた．DHT において，Ringed Bloom Filter の α をどのように設定すべきかについて，6 章に考察を記す．

一方，Fixed-size Bloom Filter を用いる手法では，全体のコンテンツ数が変化すると，最適な α の値も変

化する．全体のコンテンツ数が少ないときには，Filter ビット長が過剰となり，必要以上に FPR が小さくなる．α の値を大きく設定すると，式 (2)，(3) より FPR も大きくなる．したがって全体のコンテンツ数が少ないときには，α は大きいほうが良い．逆に，全体のコンテンツ数が多いときには，α は小さいが良い．また，本実験では，前述のとおり Fixed-size Bloom Filter の Filter サイズ m を，コンテンツ数が 50,000 のときに最もトラフィック量を削減できる値に設定している．したがって，コンテンツ数が 50,000 より増えると，トラフィック量も急激に増加している．だが，α や Filter サイズ m の値は，システム全体のノードで共通の値を使用する必要があるので，全体のコンテンツ数に応じて値を変更することは，分散環境では困難である．

図 8 において，Fixed-size Bloom Filter を用いた手法では，平均すると，$\alpha = 1/2^5$ のときが最もトラフィック量を削減できた．Ringed Bloom Filter を用いた手法に関しては，$\alpha = 1/2^7$ のときが最もトラフィック量を削減できた．このとき，Ringed Bloom Filter を用いた手法は，Fixed-size Bloom Filter を用いた手法よりも，平均トラフィック量を約 39%している．また，図 9 より，最大トラフィック量を約 73%削減したことが分かる．

6. 考　　察

6.1 α の設定

α はシステム全体のノードであらかじめ共通に設定する必要がある．ここでは，Filter を送る側のノードと受け取る側のノードがそれぞれ同じコンテンツ数 C を保持しているという状況において，検索時のトラフィック量を最も削減できる FPR の値を求める．

ユーザによる 2 語の検索語を，w_1，w_2 とする．それぞれの単語の担当ノードを N_{w_1}，N_{w_2} とする．単語 w_i で登録されているコンテンツの集合を C_i と表し，この集合から作成された Ringed Bloom Filter を $B[C_i]$ と表す．

ユーザから検索語 w_1&w_2 を受けた N_{w_1} は，$B[C_1]$ を作成し，これを N_{w_2} に送信する．N_{w_2} は，$C_1 \cap C_2$ を抽出するが，False positive により誤ったコンテンツも抽出される．これらを合わせたものを，$B[C_1] \cap C_2$ とおく．False Positive の結果を取り除くため，$B[C_1] \cap C_2$ を再び N_{w_1} に送信する．

コンテンツ ID を表すビット数を ζ とし，FPR を α とする．N_{w_1} で作成される Ringed Bloom Filter のサイズは，式 (7) より $|B[C_1]| = \lfloor \lfloor |C_1| \cdot \log_{1/2} \alpha \rfloor / \ln 2 \rfloor$ となる．整数演算を取り除くと，$|B[C_1]| = -|C_1| \cdot \ln \alpha / (\ln 2)^2$ となる．また，$|B[C_1] \cap C_2| = |C_1 \cap C_2| + \alpha \cdot (|C_2| - |C_1 \cap C_2|)$ である．

N_{w_1} と N_{w_2} 間で送信されるトラフィック量の合計は，

$$T(w_1, w_2) = |B[C_1]| + \zeta \cdot |B[C_1] \cap C_2| \quad (17)$$

となる．したがって，

$$\alpha = \frac{|C_1|}{\zeta \cdot (\ln 2)^2 \cdot (|C_2| - |C_1 \cap C_2|)} \quad (18)$$

のとき，式 (17) は最小値をとる．

また，多くの場合 $|C_2| \gg |C_1 \cap C_2|$ であると仮定すると，

$$\alpha \approx \frac{|C_1|}{\zeta \cdot (\ln 2)^2 \cdot |C_2|} \quad (19)$$

と表すことができる．また，コンテンツ ID のビット数 $\zeta = 160$，$C = |C_1| = |C_2|$ である場合は，最適な FPR は $\alpha \approx 0.013$ となる．したがって，α を 0.013 程度に設定するときが，最もトラフィック量を削減できることになる．実際には，Filter を送る側のノードと受け取る側のノードが保持するコンテンツ ID 数が異なるため，この値がつねに最適というわけではない．Filter を送る側のノードが保持するコンテンツ ID 数が多ければ，α も大きく，逆の場合は α は小さく設定すべきであることが式 (19) から分かる．だが，平均すると，トラフィック量削減の観点においては，0.013 程度（$\approx 1/2^{6.27}$）に設定することが望ましいと考えられる．また，DHT における実験では，α を $1/2^7$ に設定したときに最もトラフィック量を削減できた．したがって，この値は妥当であると考えられる．

6.2 3 語以上の AND 検索

本論文においては，2 語の AND 検索に限定して議論を進めてきた．ここで，3 語以上の AND 検索について考察を述べる．**図 10** は，3 語による AND 検索の例を示す．検索される 3 語を w_1，w_2，w_3 とする．まず Ringed Bloom Filter を用いた TFA により，w_1 と w_2 の 2 語の AND 検索を行うことで，w_1 と w_2 を同時に含むコンテンツのコンテンツ ID（$C_1 \cap C_2$）が抽出される（図 10-1，2）．続けて，抽出されたコンテンツ ID から Ringed Bloom Filter を作成し，再び TFA を適用することで，w_1，w_2，w_3 の 3 語を同時に含むコンテンツのコンテンツ ID を抽出することができる（図 10-3，4）．このように，3 語以上の AND 検索においても，Ringed Bloom Filter を用いた手法を使用することができる．

また，w_1，w_2 を同時に含むコンテンツ数は，w_1 のみを含むコンテンツ数よりも少ない．したがって，3 語

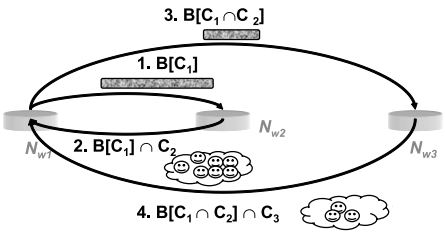


図 10 3 語以上の AND 検索
Fig. 10 Three or more words searches.

以上の AND 検索において，TFA を繰り返す際も，検索に必要なトラフィック量の増分は少ないと予想される．また，w_1，w_2 を同時に含むコンテンツ数が十分少ない場合には，TFA を繰り返さず，Naive Algorithm を併用することも考えられる．

7. おわりに

本研究では，分散ハッシュテーブルにおける，AND 検索時のトラフィック量削減に取り組んだ．分散ハッシュテーブルでは，2 語以上による AND 検索を行う場合，ノードからノードへ大量のトラフィック量が発生する．Bloom Filter を用いてこの課題に取り組む従来研究は多数あるが，Bloom Filter の持つ欠点のため，トラフィック量の削減率が制限されていた．そこで，本研究で新たに Ringed Bloom Filter を提案した．Ringed Bloom Filter が，従来の Bloom Filter よりも高い性能を持つことを示した．また，この Ringed Bloom Filter を用いることで，Bloom Filter を用いる従来手法よりも平均約 39%のトラフィック量削減に成功した．

将来課題としては，各ノードにおいて最適な False Positive Rate を設定できるような仕組みの導入が考えられる．現状では，False Positive Rate はシステム全体のノードで共通に設定する必要がある．だが本来は，ノードに登録されているコンテンツ数に応じて，ノードごとに最適な値が存在する．今後，各ノードが最適な False Positive Rate を設定できるような，新しい Filter の提案をめざす．

参考文献

1) Google. http://google.com/
2) ISO/IEC TR 15938-8:2002, Information technology, Multimedia content description interface, part 8: Extraction and use of mpeg-7 descriptions, ISO/IEC/JTC 1/SC 29 (2002).
3) Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors, *Comm. ACM*, Vol.13, No.7, pp.422–426 (1970).
4) Broder, A. and Mitzenmacher, M.: Network applications of bloom filters: A survey, *Proc. 40th Annual Allerton Conference on Communication, Control and Computing*, pp.636–646 (2002).
5) Eastlake 3rd, D. and Jones, P.: US Secure Hash Algorithm 1 (SHA1), RFC 3174 (Sep. 2001).
6) Joung, Y.-J., Fang, C.-T. and Yang, L.-W.: Keyword search in dht-based peer-to-peer networks, *Proc. 25th IEEE International Conference on Distributed Computing Systems* (*ICDCS'05*), Washington, DC, USA, pp.339–348, IEEE Computer Society (2005).
7) Li, J., Loo, B.T., Hellerstein, J.M., Kaashoek, F., Karger, D.R. and Morris, R.: On the feasibility of peer-to-peer web indexing and search, *2nd International Workshop on Peer-to-Peer Systems* (2003).
8) Miller, G.: Wordnet an on-line lexical database, International Journal of Lexicographer, Vol.3, No.4 (special issue) (1990).
9) Mullin, J.K.: Accessing textual documents using compressed indexes of arrays of small bloom filters, *Computer Journal*, Vol.30, No.4, pp.343–348 (1987).
10) Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Schenker, S.: A scalable content-addressable network, *Proc. ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, pp.161–172 (Aug. 2001).
11) Reynolds, P. and Vahdat, A.: Efficient peer-to-peer keyword searching, *Middleware*, pp.21–40 (2003).
12) Rowstron, A.I.T. and Druschel, P.: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility, *Symposium on Operating Systems Principles*, pp.188–201 (2001).
13) Sei, Y., Matsuzaki, K. and Honiden, S.: An algorithm to reduce the communication traffic for multi-word search in a distributed hash table, *Proc. 4th IFIP International Conference on Theoretical Computer Science*, pp.115–129 (2006).
14) Sei, Y., Matuzaki, K. and Honiden, S.: Reduction of the communication traffic for multi-word searches in DHTs, *Proc. International Conference on Intelligent Agents, Web Technologies and Internet Commerce* (2006).
15) Shepherd, M.A., Phillips, W.J. and Chu, C.-K.: A fixed-size bloom filter for searching textual documents, *Computer Journal*, Vol.32,

No.3, pp.212–219 (1989).
16) Stoica, I., Karger, R.D., Kaashoek, F. and Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for Internet applications, *Proc. 2001 ACM SIGCOMM Conference*, pp.149–160 (2001).
17) Tang, C., Xu, Z. and Dwarkadas, S.: Peer-to-peer information retrieval using self-organizing semantic overlay networks, *SIGCOMM*, pp.175–186 (2003).
18) Tang, C., Xu, Z. and Mahalingam, M.: pSearch: Information retrieval in structured overlays, *SIGCOMM Comput. Commun. Rev.*, Vol.33, No.1, pp.89–94 (2003).
19) Zhang, J. and Suel, T.: Efficient query evaluation on large textual collections in a peer-to-peer environment, *Peer-to-Peer Computing*, pp.225–233 (2005).

(平成 18 年 10 月 25 日受付)
(平成 19 年 4 月 6 日採録)

清　雄一

2004 年東京大学工学部システム創成学科卒業．2006 年東京大学大学院情報理工学研究科コンピュータ科学専攻修士課程修了，同年同博士課程進学，文部科学省国立情報学研究所アーキテクチャ研究系リサーチアシスタント，エージェント技術，センサネットワークの研究に従事．現在に至る．

松崎　和賢

2002 年東京大学理学部情報科学科卒業，2007 年東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程修了．同年株式会社三菱総合研究所に入社．情報理工学博士（東京大学）．

本位田真一（正会員）

1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2004 年より同研究所アーキテクチャ科学研究系研究主幹を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．2002 年 5 月〜2003 年 1 月英国 UCL ならびに Imperial College 客員研究員．2005 年度パリ第 6 大学招聘教授．早稲田大学客員教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．ソフトウェア工学，エージェント技術，ユビキタスコンピューティングの研究に従事．IEEE, ACM 等各会員，日本ソフトウェア科学会理事，情報処理学会理事を歴任．日本学術会議連携会員．