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

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．本論文では，コンテンツのキーを DHT に登録する際に 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することをめざす．本論文で提案する手法では，コンテンツのキーを DHT に登録する際に，通常保存するデータ以外のデータ保存量が発生するが，本論文で新たに Divided Bloom Filter を提案し，トラフィック量の削減とともに，新たに発生するデータ保存量の削減をめざす．

An Algorithm to Reduce the Communication Traffic for Multi-word Search in Distributed Hash Table

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

Distributed Hash Table (DHT) technology realizes Peer-to-Peer contents sharing systems. DHT system can find all contents if the contents are registered, but it lacks scalability at full text searching. In multi-word searching, there are so much communication traffic for transmission of contents IDs. In this paper, our aim is to reduce the amount of the communication traffic by using Bloom Filter. In the method, some extra storage data occur when content's key is registered to the DHT system. In this paper we propose Divided Bloom Filter and aim to reduce the amount of the communication traffic and the amount of the storage data.

1. はじめに

Peer-to-Peer システムは，中央媒介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）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，早く正確に発見することが可能となる．

DHT では，DHT システム共通で 1 つのハッシュ関数を用意し，このハッシュ関数から得られる値をもとに，コンテンツの登録・検索を行う．ノードが DHT システムに参加する際，担当すべき DHT ハッシュ値☆の範囲が与えられる．DHT ハッシュ関数がとりうる値を，ノードで分割して担当することになる．ノード内のコンテンツは，検索キーの DHT ハッシュ値を計算し，その値とコンテンツ ID，存在するノードのアドレスの組を，その DHT ハッシュ値担当ノードに登録する．ある検索キーで登録されたコンテンツが欲しい場合は，その検索キーの DHT ハッシュ値を計算し，このハッシュ値担当ノードにコンテンツの所在を尋ねる．

DHT を用いて全文検索を実現する際は，コンテンツ内の全単語が検索キーとなる．このとき，ユーザが複数の単語を同時に含むコンテンツを探すという AND 検索を行う場合に，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するという問題が起こる．文献 1) によると，トラフィック量削減の手段を講じない場合，平均 530 MB のトラフィック量が必要だと算出している．この数値が算出された環境は以下のとおりである．mit.edu [2] には 170 万の Web ページがあり，ユーザの検索履歴を解析した結果，仮にこの 170 万の Web ページが DHT で実現されているとすると，

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

☆ DHT のキー作成のためのハッシュ関数を，その他で用いるハッシュ関数と区別するため，今後この論文中では DHT ハッシュ関数と呼ぶことにし，この関数で得られる値を DHT ハッシュ値と呼ぶことにする．また，DHT ハッシュ値 x を担当するノードを x 担当ノードと呼ぶことにする．

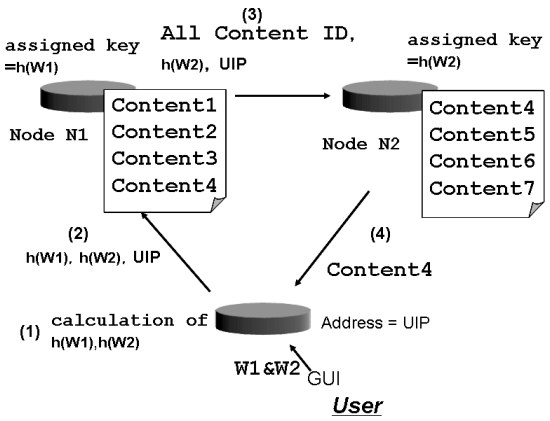


図 1 DHT における通常の AND 検索
Fig. 1 Normal multiple words searching on DHT.

1 回の AND 検索で 300 KB のデータの移動があるという結果が得られた．Google[3] では当時 30 億の Web ページをインデックスしていたので，170 万を 30 億にスケールすると，300 KB × 30 億/170 万 = 530 MB のデータの移動があることになる．たとえば，帯域幅が 1 Gbps である環境において，0.5 秒以内に検索結果を得たいとすると，最低でも AND 検索時のデータの移動を 0.5 Gb = 64 MB 以下に抑えなければならない．つまり，約 12.1%以下に削減する必要がある．

DHT における通常の AND 検索時の手順は**図 1**，**表 1**-SA のようになる．この通常の手法を Simple Algorithm（SA）と呼ぶことにする．例は，2 語の AND 検索の場合である．ノードからノードへの送信は通常複数のノードを仲介するが，本論文中では仲介役のノードの存在を省略する．

ここで，h(W1) 担当ノードから h(W2) 担当ノードへコンテンツ ID リストを送信するときに，大量の無駄なデータが発生することになる．従来研究では大きく分けて，この対策として，(1) コンテンツのキーを登録する際の工夫，(2) コンテンツ ID を転送する際の工夫，のいずれかの手法を用いている．本論文では，後述する Divided Bloom Filter（DBF）を提案し，(1), (2) 両方の工夫を行う．まず (1) として，コンテンツのキーを DHT に登録する際に Bloom Filter[4),5)] という手法を用いて，AND 検索時コンテンツ ID を送信するためのトラフィック量の削減をめざす．本論文で提案する手法では，コンテンツのキーを DHT に登録する際に，通常保存するデータ以外のデータ保存量☆が発生するが，本論文で提案する DBF を用い，トラフィック量の削減とともにこの AndSearch データ量の削減をめざす．また (2) として，コンテンツ ID リストを送信する代わりにコンテンツ ID リストの DBF を送信することでトラフィック量の削減をめざす．

以下に本論文の構成を記す．2 章では，従来研究で広く用いられ本論文でも使用する Bloom Filter や，同じ課題に取り組む従来研究を紹介する．3 章で課題に対する解決手法の提案をし，4 章で提案手法の評価・考察を行う．5 章で本論文の結論を記す．

2. 従来研究

本研究でも使用し，従来研究でもよく使われている Bloom Filter や，DHT における AND 検索時のトラフィック量削減をめざす従来研究について述べる．

2.1 Bloom Filter

Bloom Filter は，特定の集合にある要素が含まれているかどうかを，集合内の要素すべてを保持することなく判断できる，ハッシュをもとにしたデータ構造である．ここでは，Bloom Filter の概要と，Bloom Filter 適用の際の課題について述べる．

2.1.1 Bloom Filter の概要

集合 A と集合 B があり，$A \cap B$ を求める場合，単純に行うと，集合 A の全要素を集合 B に送り，集合 A，B に共通するものを取り出すことになる．このとき必要なトラフィック量は集合 A に含まれる各要素の容量の総和である．Bloom Filter を用いる手法では，集合 A 自身を送る代わりに集合 A から作成された Bloom Filter を送る．Bloom Filter のサイズは，集合 A のサイズよりも小さいので，トラフィック量の削減につながる．Bloom Filter を受け取った集合 B 側では，後述する Bloom Filter の特徴を用いることで $s_B \supseteq A \cap B$ かつ $s_B \subseteq B$ である s_B を作成することができる．ある要素が $A \cap B$ に含まれているかどうかのテストを s_B に対して行うと，ある確率で false positive（本来 $A \cap B$ には存在しない要素）が返ってくるが，false negative はけっして起こらない（$A \cap B$ に存在するのに s_B に含まれていないということはありえない）．この false positive 発生率（以下では FPR と記す）は Bloom Filter のサイズを大きくすると指数関数的に減少する．集合 B 側で作成した s_B を集合 A 側に送り返すことで，$A \cap B$ を求めることができる．

Bloom Filter の実行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

☆ 今後，この AND 検索を効率化するために各ノードに保存するデータを，AndSearch データと呼ぶ．

表 1 AND 検索の手順
Table 1 The sequence of multiple words searching.

	SA	TBFA	SBFA	STDBFA
N(Wi) が保持するコンテンツデータ	[h(Wi), コンテンツ ID, ノードアドレス] の組	SA と同じ	[h(Wi), コンテンツ ID, ノードアドレス, Bloom Filter] の組	[h(Wi), コンテンツ ID, ノードアドレス, DBF] の組
UN の処理	ユーザが指定した語 W1, W2 の DHT ハッシュ値 h(W1), h(W2) を計算	SA と同じ	SA と同じ	SA と同じ
UN → N(W1)	h(W1), h(W2), UN アドレス	SA と同じ	SA と同じ	SA と同じ
N(W1) の処理	h(W1) で登録されているコンテンツ ID の抽出	SA と同じ+抽出した ID から Bloom Filter の作成	SA と同じ+抽出した ID とともに登録されている Bloom Filter から, h(W2) を含みうる ID を抽出	SA と同じ+抽出した ID とともに登録されている DBF から, h(W2) を含みうる ID を抽出し, 抽出した ID から DBF の作成
N(W1) → N(W2)	h(W2), 抽出した ID リスト, UN アドレス	h(W2), 作成した Bloom Filter	以下 SA と同じ	h(W2), 作成した DBF
N(W2) の処理	受け取ったリストと, h(W2) で登録されているコンテンツ ID に共通で含まれている ID を抽出	h(W2) で登録されているコンテンツ ID の中から, 受け取った Bloom Filter の構成要素である可能性がある ID を抽出		h(W2) で登録されているコンテンツ ID の中から, 受け取った DBF の構成要素である可能性がある ID を抽出
N(W2) → UN	抽出した ID リスト [終了]	×		×
N(W2) → N(W1)		抽出した ID リスト		以下 TBFA と同じ
N(W1) の処理		受け取ったリストと, h(W1) で登録されているコンテンツ ID に共通で含まれている ID を抽出		
N(W1) → UN		抽出した ID リスト [終了]		

UN：ユーザノード　N(Wi)：h(Wi) 担当ノード（$i \neq j$ において $N(Wi) = N(Wj)$ となることもある）

数は $1, 2, \ldots, m$ 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 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，前述したようにある確率のもとでこの推測は間違いである．

FPR は k，m，n の関数であり，

$$\mathrm{FPR} = (1 - (1 - 1/m)^{kn})^k \quad (1)$$
$$\approx (1 - e^{-kn/m})^k \quad (2)$$

で表される[5]．式 (2) は，明らかに $k = \ln 2 \times m/n$ のとき最小値をとる．このとき，$FPR = (1/2)^k$ となる．目標 FPR を FPR_{target} とすると，k が整数値をとることを考慮して，$k = \lfloor \log_{1/2} FPR_{target} \rfloor$ となる．また，これらより，次式が導かれる．

$$m = \left\lfloor \lfloor \log_{1/2} FPR_{target} \rfloor \times n / \ln 2 \right\rfloor \quad (3)$$

2.1.2 Bloom Filter 適用の際の課題

Filter サイズ m の値は，システム全体で統一されていることが望ましい．なぜなら，Filter によって m が異なると，Filter の構成要素に含まれているかどうかをチェックするために用いるハッシュ関数も異なるので，チェックの度にハッシュ値の再計算を行う必要が出てくるからである．

だが，要素数が異なる複数の集合からそれぞれ Bloom Filter を作成するとき，m の値を統一した場合は，それぞれで最適な m を設定した場合と比較すると，全体として同じ FPR を実現したとしても，総合の Filter サイズが大きくなってしまう．これは，Filter サイズ一定のもとで集合の要素数が大きくなると指数関数的に FPR が増大するからである．

2.2 DHT における AND 検索時のトラフィック量削減

DHT における AND 検索時のトラフィック量の削減をめざす研究はいくつかあるが，大きく分けると 2 通りの手法が存在する．1 つ目は，コンテンツのキーを登録する際の工夫であり，2 つ目は，AND 検索時におけるコンテンツ ID を転送する際の工夫である．

1 つ目の手法として，文献 6) では，コンテンツに含まれるキーワードの組合せも DHT のキーとして登録している．ユーザが検索するときは平均 2.53 語の AND 検索を行うという文献 7) のデータをもとに，3 語までキーワードの組合せを作成し，DHT のキーとして登録している．だが，キーワード数が増えると組合せ数も増大するので，文献 6) では，検索対象をコンテンツのメタデータに絞り，前提としてメタデータ数は 10 以下の少数であると限定している．

また，文献 8) では，Resource Description Framework [9]（RDF）ドキュメントを対象にし，RDF triple を DHT で分散させて保持するシステムを構築している．ここでは，triple 内の AND 検索時におけるトラフィック量を削減するために，triple 内のそれぞれのキーワードとともに，triple そのものも登録している．triple 内の要素は 3 つしか存在しないので，AndSearch データ量は少量に抑えられているが，全文検索等，DHT のキーとなる要素がコンテンツに大量に含まれているときは，AndSearch データ量が膨大になってしまう．文献 10) では，コンテンツの内容を要約して DHT のキーとして登録している．キーの数自体を削減しているので，AND 検索時のトラフィック量の削減につながっている．だが，要約することでコンテンツに含まれる情報量も削減されており，本研究がめざす全文検索には適用することができない．

2 つ目の手法として，文献 1)，7)，11) では，AND 検索におけるコンテンツ ID 転送時に Bloom Filter という手法を用いている．コンテンツ ID 転送時に Bloom Filter を使用することで，トラフィック量の削減が可能となる．今後，この手法を Transmission Bloom Filter Algorithm（TBFA）と呼ぶ．

TBFA を用いた具体的な手順は表 1-TBFA のようになる．ノード N(W2) からノード N(W1) へ，抽出した ID リストを送信しているのは，false positive の発生により含まれてしまった ID を削除するためである．

これらコンテンツ ID 送信時に Bloom Filter を使う手法の長所は，AndSearch データ量が発生しないところだが，トラフィック量の削減率が 1 つ目の手法に比べると小さいという問題がある．

3. 提 案 手 法

前章で紹介した Bloom Filter を，本研究では従来研究と異なる部分で使用する．従来研究ではコンテンツ ID 転送時に Bloom Filter を使用しているが，本研究ではコンテンツのキーワード登録時に使用する．このとき発生する課題とその取り組みについても述べる．

3.1 Saving Bloom Filter Algorithm（SBFA）

本論文で提案する手法は，コンテンツのキー登録時の工夫である．コンテンツ内に含まれる単語を担当ノードに登録する際に，コンテンツに含まれるその他すべての単語についての Bloom Filter もともに登録する．登録の際の具体的な手順は次のようになる．

(1) 登録する単語 W1 以外の単語について，DHT ハッシュ値を計算する．
(2) (1) で計算したすべての DHT ハッシュ値の Bloom Filter を作成する．
(3) h(W1) 担当ノードに，コンテンツ ID を登録する際，(2) で計算した Bloom Filter もともに登録する．

この手法を Saving Bloom Filter Algorithm（SBFA）と呼ぶことにする．AND 検索時の手順は表 1-SBFA のようになる．

3.2 SBFA おける課題

2.1 節で述べたように，最適な Filter サイズは，集合の要素数に依存する．本論文中では，集合の要素数とはコンテンツに含まれる単語の数のことである．コンテンツに含まれる単語数には偏りがあるので，最適な Filter サイズを決定できないという問題がある．Filter を作成するときに用いる k 個のハッシュ関数は DHT システム全体で統一しておく必要があるので，Filter のサイズもシステム全体で共通でなければならない．

予想される範囲内で，十分大きな Filter サイズを設定することも可能であるが，データ保存量やトラフィック量が増加してしまうことになる．逆に，Filter のサイズを小さくし過ぎた場合，FPR が上昇するのでトラフィック量が増加してしまう．

3.3 Saving Divided Bloom Filter Algorithm（SDBFA）

ある FPR のもとで Filter サイズを一定にするには，Filter の構成要素数を一定にする必要がある．

式 (3) より，集合の要素数 n で，ある FPR_{target} を満たすために必要な Filter ビット数 m は n とほぼ比例関係にある．この性質を利用し，Filter を作成するときに，集合を一定の構成要素数に分割しそれぞれで

Filterを作成することで，FPRを上昇させることなしに各Filterサイズを一定に保つことができる．分割して作成したフィルタをDivided Bloom Filter（DBF）と呼ぶことにし，DBFをコンテンツIDとともに担当ノードに登録する手法をSaving Divided Bloom Filter Algorithm（SDBFA）と呼ぶ．この場合，分割したそれぞれのFilterにおいて，FPR $= \alpha$ となる．

このとき，以下の問題が生じる．ある要素bがDBFの構成要素であるかどうかを判定するときに，分割したそれぞれのフィルタについてチェックを行うと，分割したFilter数を M とした場合

$$\mathrm{FPR} = 1 - (1-\alpha)^M \quad (4)$$

α が十分小さいとすると，α の二乗以上の項は無視できるので

$$\mathrm{FPR} \fallingdotseq M \times \alpha \quad (5)$$

となり，分割数に応じてFPRが増大してしまう．

解決策として，分割したFilterのうち，要素bが含まれうるFilterをただ1つ特定することができればよい．この場合FPRは全体として α となる．以下で，AndSearchデータ量を増やすことなく，このFilterを特定する手法を述べる．

コンテンツに含まれる単語の集合を分割する際，各単語を，DHTハッシュ関数でハッシュ値を計算し，このハッシュ値に基づいて，グループ分けを行うようにする．

このために，システムとして，以下を決めておく．

- 1つのグループに含む単語の平均数 MN
- 目標の FPR_{target}

MN と FPR_{target} より，分割された各Filterのサイズは，式(3)を用いて決定される．したがって，コンテンツに含まれる単語数 n にかかわらず，各Filterサイズを決定することができる．コンテンツCに含まれる単語をグループ分けする際の，具体的な手順は次のようになる．DHTハッシュ値がとりうる値を $1, 2, \ldots, DN-1$ とする．

(1) コンテンツCに含まれる単語数 WN に応じて，グループの数
$$GN = \lfloor WN/MN + 0.5 \rfloor \quad (6)$$
を計算する．

(2) 各グループ G_i $(i = 1, \ldots, GN)$ に，担当ハッシュ範囲 $R(G_i) = \big[(DN/GN) \times (i-1), (DN/GN) \times i\big)$ を与える．

(3) コンテンツCに含まれる単語を1つ取り出してwとし，DHTハッシュ値h(w)を計算する．

(4) ある $R(G_j)$ がh(w)を含むなら，wをグループ G_j に挿入する．

(5) 上記(3)–(4)をコンテンツCに含まれるすべての単語に対して行う．

式(3)とあわせて，DBFのAndSearchデータ量は，

$$m = \lfloor WN/MN + 0.5 \rfloor \times \lfloor k \times MN/\ln 2 \rfloor \quad (7)$$

となる．DHTハッシュ関数としてハッシュ値に値域上の偏りがないハッシュ関数を用いていれば，すべてのグループにほぼ同数の単語が振り分けられることになる．

ある単語 b がこのコンテンツCに含まれているかどうかの判断は次のような手順で行う．

(1) Filterの数から，それぞれのグループ G_i の担当ハッシュ範囲 $R(G_i)$ を前述した計算式で求める．

(2) 単語 b のDHTハッシュ値h(b)を計算する．

(3) h(b)を含む $R(G_j)$ を特定する．このとき，b はグループ G_j のみの構成要素になりうる．

(4) 単語 b が，グループ G_j から作られたFilterの構成要素に含まれている可能性があるかどうかを判断する．この判断がそのまま，単語 b がコンテンツCに含まれている可能性があるかどうかの判断になる．

このSDBFAを用いることで，コンテンツに含まれる単語数に偏りがあったとしても，目標とするFPRを満たすためのより最小に近いサイズのFilterを与えることができる．

3.4 Saving and Transmission Divided Bloom Filter Algorithm（STDBFA）

DBFを用いる手法は，コンテンツ登録時だけではなく，コンテンツID送信時にも用いることができる．具体的には，表1-TBFA内における，ノードN(W1)の処理の部分で，Bloom Filterではなく DBFを作成する．これにより，全体としてAndSearchデータ量を減らしたうえで，同等のFPRを実現することが可能となる．

SDBFAと，コンテンツID送信時にDBFを用いる手法をあわせたアルゴリズムを，Saving and Transmission Divided Bloom Filter Algorithm（STDBFA）と呼ぶことにする．STDBFAにおけるAND検索時の手順を表1-STDBFAに示す．

4. 評　　価

本論文中で紹介した，SA，TBFA，SBFA，SDBFA，STDBFAについて実験をし評価を行う．このうち，SAはトラフィック量削減の対策を何も講じない手法であり，TBFAは従来研究で用いられている手法である．SBFA，SDBFA，STDBFAは本研究で提案する手法

である．

実験は，上記 5 つの手法について，平均トラフィック量を測定する．また，それぞれの手法に必要な，AndSearch データ量についても比較する．

4.1 実験設定

1 章で述べたように，AND 検索時のトラフィック量を 12.1%以下に抑えることを目標とする．本手法は，この条件より緩い範囲において有効であることを示す．

実験に用いるコンテンツとして，英語論文を 1 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 [12] 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．論文に含まれる平均単語種類数は 831 語であった．

ノード数は 10,000 とし，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 [13] 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ことになる．

トラフィック量の計算に関して，コンテンツ ID 送信時に Bloom Filter または DBF を使う TBFA, STDBFA については，以下のように行う．表 1-TBFA, STDBFA において，ノード N(W1) からノード N(W2) へのトラフィック量と，ノード N(W2) からノード N(W1) へのトラフィック量の合計を，AND 検索時に必要なトラフィック量とする．そのほか，SA, SBFA, SDBFA については，ノード N(W1) からノード N(W2) へのトラフィック量を AND 検索時に必要なトラフィック量とする．

SBFA や TBFA は，従来の Bloom Filter を用いる手法である．従来手法では，計算量を削減するために Filter サイズをあらかじめ 1 つに決定して使用する．ここでは，対象となる集合の要素数の平均値をもとに，Filter サイズを決定した．SBFA については，実験に用いる 1 万コンテンツの平均単語保持数が 831 語であったため，Filter サイズを，式 (3) を用いて，集合の要素数 n が 831 のもとで，指定した FPR を満たす値に設定した．TBFA では，実験に用いるコンテンツの単語 1 つあたりのコンテンツ登録の平均数が約 197 であったため，コンテンツ ID を送信する際に作成する Filter のサイズを，式 (3) を用いて，集合の要素数 n が 197 のもとで，指定した FPR を満たす値に設定した．

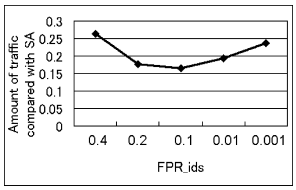


図 2 TBFA の実験結果（トラフィック量）
Fig. 2 The result of TBFA (Amount of traffic).

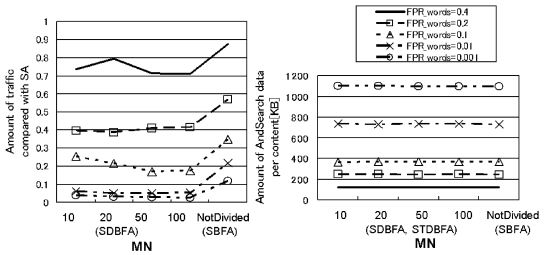


図 3 SBFA と SDBFA の実験結果
左：SA と比較したトラフィック量，右：AndSearch データ量
Fig. 3 The result of SBFA and SDBFA.
Left: Amount of traffic compared with SA, Right: Amount of AndSearch data.

4.2 実験結果

実験はすべて 1,000 回の試行を行い，その平均値をとった．それぞれにおける AndSearch データ量は，実際にノードに保存されたデータ量を計算した．また，SA のトラフィック量は平均 2.97 KB であった．

また，以下で，コンテンツに含まれる単語から作成する Filter の FPR_{target} を FPR_{words}，コンテンツ ID から作成する Filter の FPR_{target} を FPR_{ids} と記述する．

従来研究である TBFA では，$FPR_{ids} = 0.4$, 0.2, 0.1, 0.01, 0.001 について実験を行った（**図 2**）．結果より，$FPR_{ids} = 0.1$ のとき最もトラフィック量が少なく，このときのトラフィック量は SA と比較して 17%であった．設定する FPR が小さいと，表 1-TBFA において，ノード N(W1) からノード N(W2) へ送信する Filter サイズが大きくなる．逆に FPR が大きいと，ノード N(W2) からノード N(W1) へ送信するコンテンツ ID の数が多くなる．

SBFA では，$FPR_{words} = 0.4$, 0.2, 0.1, 0.01, 0.001 について実験を行った（**図 3** 左図，右図，それぞれ右端）．図 3 左図は，AND 検索時のトラフィック量を，図 3 右図は，1 つのコンテンツを DHT に登録するときに必要な AndSearch データ量，つまり，Bloom Filter の総合データ量を表している．FPR を小さく設定すると AndSearch データ量が増大している．これは，式 (3) を満たしている．

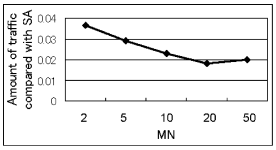


図 4 STDBFA の実験結果（トラフィック量）
Fig. 4 The result of STDBFA (Amount of traffic).

SDBFA では，FPR_{words} を SBFA と同様に設定し，それぞれにおいて，$MN = 10$，20，50，100 について実験を行った（図 3 左図，右図，それぞれ右端以外．AndSearch データ量に関しては，SDBFA と STDBFA は共通）．図 3 左図によると，通常の Bloom Filter を用いる SBFA よりも DBF を用いる SDBFA のほうがトラフィック量を削減できていることが分かる．また，図 3 右図によると，通常の Bloom Filter を用いた場合と，DBF を用いた場合では，AndSearch データ量にあまり違いはなかった．式 (7) からも，AndSearch データ量は MN にあまり依存しないことが分かる．また，$FPR_{words} = 0.01$ のとき，目標の 12.1%を達成できている．

次に実験を行う STDBFA では，コンテンツのキー登録時には $FPR_{words} = 0.01$，$MN = 10$ で DBF を作成し，AND 検索時には $FPR_{ids} = 0.1$，$MN = 2$，5，10，20，50 について実験を行った（**図 4**）．図 4 より，AND 検索時には $MN = 20$ のときが最もトラフィック量を削減できていることが分かる．MN が小さいとトラフィック量が増大する理由は以下のとおりである．要素をグループ分けする際に各グループには平均 MN の要素が振り分けられるが，MN が小さいと，[(グループ内の実際の要素数)/MN] が大きいグループが出てくる．要素数が増えると，Filter の FPR は指数関数的に増大するので，全体として DBF の FPR が増大してしまう．逆に MN が大きいとトラフィック量が増大する理由は，集合の要素を分割するときに生じる端数が大きくなるためである．

ここまでの実験はコンテンツ数 10,000 において実験を行った．次に，コンテンツ数を 1,000 から 10,000 まで増やしたときの，トラフィック量の推移を測った（**図 5**）．各 1,000 回ずつ試行し，図はこの平均値を示してある．各測定において，ユーザが検索する語は異なるので，結果にバラつきが生じている．だが従来手法である TBFA と比較すると，提案手法である SDBFA や STDBFA は，安定して良好な結果を残していることが分かる．

これら実験結果をまとめたものを，**表 2** に示す．これはコンテンツ数を 1,000 から 10,000 まで増やしたときの，平均値をとってある．従来研究の手法である TBFA では，AND 検索時のトラフィック量は SA と比較して 23.7%必要だが，コンテンツのキー登録時にコンテンツに含まれる単語の Bloom Filter もともに登録するという SBFA を用いることで，TBFA よりもトラフィック量の削減に成功している．また，SDBFA や STDBFA では，通常の Bloom Filter ではなく本研究で提案した DBF を用いることで，SBFA よりもトラフィック量・AndSearch データ保存量ともに削減することができた．

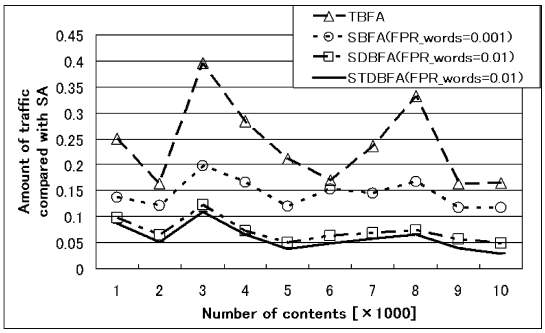


図 5 コンテンツ数を変化させたときの各トラフィック量
Fig. 5 Amount of traffic with change in the number of contents.

表 2 実験結果の比較
Table 2 The comparison of the result.

	Amount of traffic compared with SA	Amount of data storage per content [KB]
TBFA	0.237	
SBFA	0.144	1095
Desired Value	0.121	
SDBFA	0.072	730
STDBFA	0.059	730

4.3 考　　察

現在では一般に，動画や音楽等のマルチメディアコンテンツにはテキストデータは付随していない．だが，MPEG7[14] 等のメタデータ記述言語によりマルチメディアコンテンツにテキストを埋め込むことが可能である．マルチメディアコンテンツへのメタデータ記述を，将来的に人手ではなく自動で付与できるようになると，コンテンツに大量のメタデータが付与されると考えられる．このマルチメディアコンテンツを対象に DHT システムを構築すると，AND 検索時のトラフィック量も増大する．だが，本研究で提案した手法を用いることで，このトラフィック量を削減することができる．

また，本研究では 2 語の AND 検索に限定して評価を行った．3 語の AND 検索を行う場合の考察を記

す．3 語を W1，W2，W3 とし，h(W1) 担当ノードを N(W1) とする．SA，TBFA に関しては，送信すべきトラフィック量の最大値は増減しない．だが SBFA，SDBFA，STDBFA に関しては，ノード N(W1) において，登録されている Bloom Filter や DBF から，W2 を含みえ，かつ，W3 を含みうるコンテンツ ID のみを抽出するので，2 語の AND 検索を行う場合よりもトラフィック量が減少すると考えられる．

次に，コンテンツ数が増えるときの提案手法の有効性について議論する．コンテンツ数が N 倍になるとき，(SDBFA/SA) のトラフィック量の比が何倍になるかを計算する．この比が 1 以下ならば，コンテンツが増えるほど，SA に対する SDBFA のトラフィック量の比は減少し，より良い結果を得られることになる．コンテンツ数が N 倍になると，すべてのコンテンツを通しての単語種類数の平均が $f(N)$ 倍になるとする．このとき，$1 < f(N) < N$ を満たすと考えられる．また，1 つの単語あたりのコンテンツ登録数は，$Sa = N/f(N)$ 倍となる．したがって，SA のトラフィック量は Sa 倍になる．ある 2 語を両方とも含むコンテンツ数は，同様の議論により，$Sa/f(Sa)$ 倍となる．これにより，SDBFA のトラフィック量は，FPR_{words} を α とすると，$Sd = Sa/f(Sa) + (Sa - Sa/f(Sa)) \times \alpha$ 倍となる．したがって，$1 < f(Sa)$ と $0 < \alpha < 1$ を考慮すると，$Sd/Sa < 1$ となる．結果，コンテンツ数が増えるほど，SA に対する SDBFA のトラフィック量の比は減少し，提案手法の有効性は増すと考えられる．

5. おわりに

本論文では，DHT の AND 検索時，コンテンツ ID リストを送信する際に発生する，トラフィック量の削減に取り組んだ．まず，(1) コンテンツのキーを登録する際の工夫として，コンテンツのキー登録時にコンテンツに含まれる単語の Bloom Filter もともに登録することで，トラフィック量の削減が可能となった．この手法では，コンテンツのキーを DHT に登録する際に，AndSearch データ量が発生するが，本論文で新たに Divided Bloom Filter を提案し，トラフィック量の削減とともに AndSearch データ量の削減に取り組んだ．また，(2) コンテンツ ID を送信する際の工夫として，コンテンツ ID リストを送信する代わりにコンテンツ ID リストの DBF を送信することで，トラフィック量の削減に取り組んだ．

DBF は，通常の Bloom Filter での課題であった，集合サイズのばらつきに対応するために提案した．DBF では，集合サイズにばらつきがあっても，指定した False Positive Rate を満たす最小に近いサイズの Filter を作成することが可能となった．

本研究の提案手法となる STDBFA では，DBF を用いることで，DHT の AND 検索時のトラフィック量削減とともに，AndSearch データ量に対して良好な結果を得ることができた．

参考文献

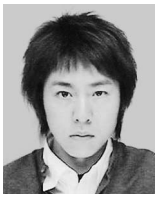
1) Li, J., Loo, B.T., Hellerstein, J.M. and Kaashoek, M.F.: On the Feasibility of Peer-to-Peer Web Indexing and Search, *International Workshop on Peer-To-Peer Systems* (2003).
2) Massachusetts Institute of Technology. http://mit.edu/
3) Google. http://google.com/
4) Bloom, B.: Space/time trade-offs in hash coding with allowable errors, *Comm. ACM*, Vol.13, No.7, pp.422–426 (1970).
5) Broder, A. and Mitzenmacher, M.: Network applications of bloom filters: A survey, *Annual Allerton Conf. on Communication, Control and Computing*, pp.636–646 (2002).
6) Clements, A.T., Ports, D.R.K. and Karger, D.R.: Arpeggio: Metadata Searching and Content Sharing with Chord, *International Workshop on Peer-To-Peer Systems* (2005).
7) Reynolds, P. and Vahdat, A.: Efficient Peer-to-Peer Keyword Searching, *Middleware Conference* (2003).
8) Cai, M. and Frank, M.: RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network, *International conference on World Wide Web* (2004).
9) http://www.w3.org/RDF, World-Wide Web Consortium: Resource Description Framework.
10) Tang, C., Xu, Z. and Dwarkadas, S.: Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks, *Special Interest Group on Data Communications*, Karlsruhe, Germany (Aug. 2003).
11) Zhang, J. and Suel, T.: Efficient Query Evaluation on Large Textual Collections in a Peer-to-Peer Environment, *International Conference on Peer-to-Peer Computing* (2005).
12) WordNet. http://wordnet.princeton.edu/
13) Eastlake, D. and Jones, P.: US Secure Hash Algorithm 1 (SHA1), IETF Request for Comments 3174 (2001).
14) ISO/IEC TR 15938-8:2002: Information technology. Multimedia content description interface. Part8: Extraction and use of MPEG-7

descriptions, ISO/IEC/JTC 1/SC 29 (2002).

(平成 17 年 10 月 4 日受付)
(平成 18 年 3 月 2 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知能システム研究系リサーチアシスタント，エージェント技術の研究に従事．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．2004 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知能システム研究系リサーチアシスタント，エージェント技術の研究に従事．現在に至る．日本ソフトウェア科学会学生会員．

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併任，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（文部科学省在外研究員）．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，日本ソフトウェア科学会等各会員．本学会理事．