Vol.2011-DPS-146 No.27
Vol.2011-CSEC-52 No.27
2011/3/11

P2P ネットワークにおける Skip Graph と Bloom Filter を用いた効率的な複数キーワード検索手法の提案

岩本大記[†1]　安倍広多[†1]
石橋勇人[†1]　松浦敏雄[†1]

構造化 P2P ネットワークとして多く用いられている分散ハッシュテーブルでは，1 つの key に対して 1 つの value を対応づけ，非常に多数のノードに分散して格納する．このため，複数のキーワードを同時に含むデータを効率良く検索することは困難である．この問題を解決するため，本稿では，Bloom filter を利用した検索手法を提案する．Bloom filter を用いることによって，検索対象のデータが含むすべてのキーワードをコンパクトに記憶することが可能となる．各ノードが持つ Bloom filter の内容は集約 Skip graph に基づくアルゴリズムによって集約され，保持される．検索時にはそれを用いて対象となるノードの範囲を絞り込んでゆくため，複数キーワードによる検索を効率良く実行することが可能である．提案手法の有効性を確認するため，シミュレーションによる実験をおこなったので，その結果についても述べる．

A Proposal of an Efficient Multiple Keywords Search Method on P2P Network using Skip Graph and Bloom Filter

Iwamoto Taiki,[†1] Kota Abe,[†1] Hayato Ishibashi[†1]
and Toshio Matsuura[†1]

DHT (Distributed Hash Table) is a popular data structure for structured P2P networks. A DHT is a distributed key-value store that manages mapping from keys to values by means that one key corresponds to one value. Due to this property, it is difficult to perform multiple keyword search efficiently. This paper proposes an efficient search method using *Bloom filter*, a space-efficient data structure storing multiple keys in one bit sequence. Bloom filters on a series of nodes are aggregated by an algorithm based on *Aggregation Skip Graph* so that narrowing search ranges can be efficient in time. Simulation results are also reported.

1. はじめに

インターネット上でのサービスを提供する手法として，耐故障性やスケーラビリティの点から Peer-to-Peer(P2P) 技術が注目されている．P2P ネットワークの分野では，とくに分散ハッシュテーブル (DHT) に基づく構造化 P2P ネットワークがよく研究されている．DHT では，key と value のペアを P2P ネットワーク上のノードに分散配置し，key を指定することで対応する value を効率良く発見できる．

しかし，DHT ではネットワーク上のノード上に分散したドキュメントに対して全文検索を行うことは難しい．これは DHT では複数キーワードによる絞り込み（AND 検索）を効率的に実現できないためである．

本稿では，Bloom filter[1] と集約 Skip graph[2] を用いることで，複数キーワードによる AND 検索を効率的に行う方式（**BF Skip graph** と呼ぶ）を提案する．BF Skip graph では，複数キーワードによる AND 検索を，キーワードの数によらず，$O(\log N)$ 時間で実行できる（N はノード数）．

以下，2 章で BF Skip graph が利用するデータ構造について簡単に述べ，3 章で BF Skip graph のアルゴリズムを述べる．4 章でシミュレーションによる評価を行い，最後に 5 章でまとめと今後の課題を述べる．

2. 準　　備

ここでは，提案手法で用いる Skip graph と集約 Skip graph，および Bloom filter について簡単に説明する．

2.1 Skip graph

Skip graph[3] は構造化オーバレイネットワークの一つであり，P2P システムに適した分散データ構造である．Skip graph では，各ノードは key と呼ばれる値を保持する．また，各ノードには membership vector と呼ばれる基数 w の整数値が乱数により割り当てられる．

†1 大阪市立大学大学院創造都市研究科
Graduate School for Creative Cities, Osaka City University

Vol.2011-DPS-146 No.27
Vol.2011-CSEC-52 No.27
2011/3/11

Skip graph には複数のレベルがあり，レベル l では，membership vector のプレフィックスの l 桁が一致するノード同士で双方向リンクを張ることで，key の昇順に並んだ双方向連結リストを構成する（すべてのノードはレベル 0 の連結リストで繋がっている）．レベル l では高々 w^l 個の連結リストが存在し，各連結リストに属するノード数は平均するとレベル $(l-1)$ でのノード数の $1/w$ となる．

なお，本稿では連結リストの左端と右端のノードは繋がっているものとする．また，各ノードで連結リスト上に存在するノード数が 1 となるレベルを maxlevel と呼ぶ．N ノードからなる Skip graph では，maxlevel は平均 $O(\log_w N)$ である．また，ノードの検索，挿入に要するメッセージ数およびホップ数はともに $O(w \log_w N)$ である．

2.2 集約 Skip graph

複数のノードが保持する値に対して，最大値や最小値，平均や合計といった集約値を求めるクエリを集約クエリと呼ぶ．集約クエリを効率良く実行できるデータ構造として，Skip graph を拡張した集約 Skip graph[2] がある．集約 Skip graph の基本的な構造は上で述べた Skip graph と同じである．集約 Skip graph では，各ノードは key と value のペアでデータを保持する．ノードは Skip graph と同様に key の昇順でソートされる．

集約 Skip graph のノード p は，各レベルにおいて，p から p の右ノードまでの範囲のノードが保持する value の集約値を保持する（集約値は定期的に更新する）．この集約値を利用することで，指定した key の範囲のノードが保持する value に関する集約クエリを $O(\log N + \log r)$ 時間で実行できる（N は全ノード数，n は指定した範囲内に含まれるノード数）．

2.3 Bloom filter

Bloom filter[1] は，複数の要素からなる集合の中に目的とする要素が含まれているかどうかを定数時間で判定可能なデータ構造である．Bloom filter はサイズが $m(m>0)$ のビット列と 0 から $m-1$ までの値を返す $k(k>0)$ 個のハッシュ関数で構成される．Bloom filter に要素 u を追加する場合，u から求めた k 個のハッシュ値に対応する Bloom filter のインデックスの各ビットの値をセットする．ある要素 v が Bloom filter に含まれているかの判定は，要素を追加する場合と同様のハッシュ関数を用いて，Bloom filter の k 個のインデックスのビットの値をチェックすることで行なう．この k 個のビットの 1 つでも “0” があれば，要素 v は Bloom filter に含まれていないと判断できる．k 個の全てのビットが “1” の場合，要素 v は Bloom filter に含まれていると判断するが，ハッシュ値の衝突が起こる可能性があるため，この判断は間違っている可能性がある（これを**偽陽性**と呼ぶ）．

集合 S_1 に対応する Bloom filter b_1 と，集合 S_2 に対応する Bloom filter b_2 があるとき，b_1 と b_2 のビット単位 OR を取ることで集合 $S_1 \cup S_2$ に対応する Bloom filter が得られる（ただし，ハッシュ関数群とビット数 m は同一である必要がある）．

3. 提案手法

BF Skip graph は，ネットワーク上に分散するノードが保持する多数のドキュメントから，すべての指定したキーワードを含むドキュメントを効率良く検索する（AND 検索）ための構造化 P2P ネットワークである．

3.1 概　　要

BF Skip graph は Skip graph および集約 Skip graph をベースとしている．BF Skip graph に参加するノード n は Skip graph と同様の手順で BF Skip graph に挿入する．

Skip graph において，ノード n のレベル l における左右のノードをそれぞれ $n.\mathrm{left}[l]$, $n.\mathrm{right}[l]$ で表す．また，同一の連結リスト上に存在するノード n, m に対し，区間 $[n,m)_l$ は n および m が属するレベル l の連結リスト上で，n から m までの間に存在するノード集合（ただし m は含まない）を意味するものとする．また，区間 $(n,m)_l$ は，n, m をともに含まない集合である．

BF Skip graph では，ノード n は各レベル l で Bloom filter の集合 $n.\mathrm{bf}[l]$ を保持する．$n.\mathrm{bf}[l]$ は，集約 Skip graph における各レベルの集約値に相当する．$n.\mathrm{bf}[0]$ は，n 自身が保持する各ドキュメント $(\mathrm{d}_1, \mathrm{d}_2, \ldots)$ を Bloom filter に変換したものを格納する $(n.\mathrm{bf}[0] = \{\mathrm{bf}(\mathrm{d}_1), \mathrm{bf}(\mathrm{d}_2), \ldots\})$．ただし bf() はドキュメントを Bloom filter に変換する関数である．

また，$n.\mathrm{bf}[l]$ $(l>0)$ は，すべてのノード $x \in (n, n.\mathrm{right}[l])_{l-1}$ から取得した Bloom filter を格納する．ここで，ノード n が x から取得する Bloom filter は，$x.\mathrm{bf}[0]$ から $x.\mathrm{bf}[l-1]$ (以下 $x.\mathrm{bf}[0 \ldots l-1]$ で表す）に含まれるすべての Bloom filter のビット単位 OR を取ったもので，以下 $\mathrm{BOR}(x.\mathrm{bf}, l-1)$ で表す．

ここで，$\mathrm{BOR}(x.\mathrm{bf}, l-1)$ は，区間 $[x, x.\mathrm{right}[l-1])_o$ 内のノードが保持するドキュメントに対応する Bloom filter である．区間 $[n, n.\mathrm{right}[l-1])_0$ の範囲の Bloom filter は，$\mathrm{BOR}(n.\mathrm{bf}, l-1)$ で得られるため，結局 $\mathrm{BOR}(n.\mathrm{bf}, l)$ によって $[n, n.\mathrm{right}[l])_0$ の範囲のドキュメントに対応する Bloom filter が得られる（**図 1** 参照）．また，$\mathrm{BOR}(n.\mathrm{bf}, \mathrm{maxlevel})$ はすべてのノードが保持するドキュメントに対応する Bloom filter である．

なお，$n.\mathrm{bf}$ に格納する Bloom filter b には，n が b を取得したノードへのポインタ情報

Vol.2011-DPS-146 No.27
Vol.2011-CSEC-52 No.27
2011/3/11

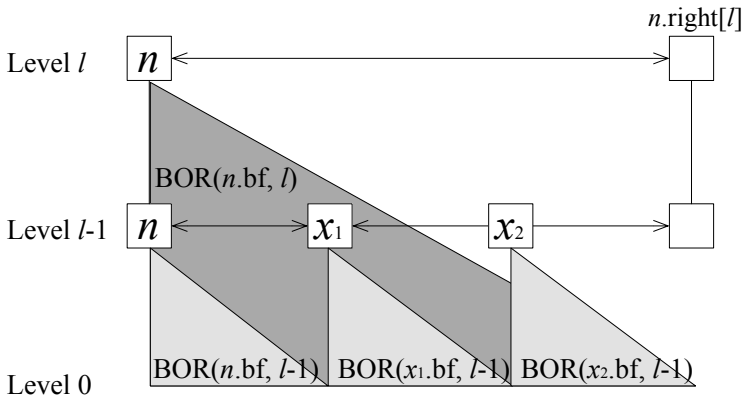

図 1 ノードが各レベルで保持する Bloom filter と BOR との関係

($b.ptr$ で表す）が付与されている．n が Bloom filter b を x から取得した場合 $b.ptr = x$ である．また，ノード n がレベル 0 で保持する Bloom filter $b(\in n.\mathrm{bf}[0]))$ では，$b.ptr = n$ とし，さらに，b に対応するドキュメントの情報（ファイルの内容など）$b.doc$ が付与されている．

BF Skip graph において各ノードが保持するデータを**表 1** に示す．

Skip graph では，ノードは key によってソートされる．BF Skip graph でも同様であるが，BF Skip graph ではノードの並び順は重要ではないため，key の値は IP アドレスや乱数など，任意の値を用いることができる．以下，key が u のノードを N_u と表記する．

BF Skip graph の例を**図 2** に示す．図中の正方形の中の数字は key を表す．

ノード数を N とするとき，maxlevel は $O(\log_w N)$ であり，また各レベルで保持する Bloom filter の数は $O(w)$ であるため，ノードが保持する Bloom filter の総数は，$O(w \log_w N)$ となる．

キーワード検索は最上位レベル (maxlevel) から行う．ノード n がレベル l でキーワード検索要求を受信した場合，$n.\mathrm{bf}[0 \ldots l-1]$ から指定されたキーワードが含まれる可能性のある Bloom filter を全て検索する．マッチする Bloom filter がレベル $l > 0$ で見つかった場合，その Bloom filter の取得元ノードへ検索要求を転送する（マッチする全ての Bloom filter の取得元ノードに転送する）．レベル 0 で見つかった場合は当該ノードが検索要求にマッチするドキュメントを保持している．

表 1 BF Skip graph で各ノードが保持するデータ

変数	説明
key	キー
m	membership vector
left[*level*]	各レベルでの左ノードへのポインタ
right[*level*]	各レベルでの右ノードへのポインタ
maxlevel	連結リスト上でノード数が 1 となるレベル
bf[*level*]	各レベルでの Bloom filter の集合

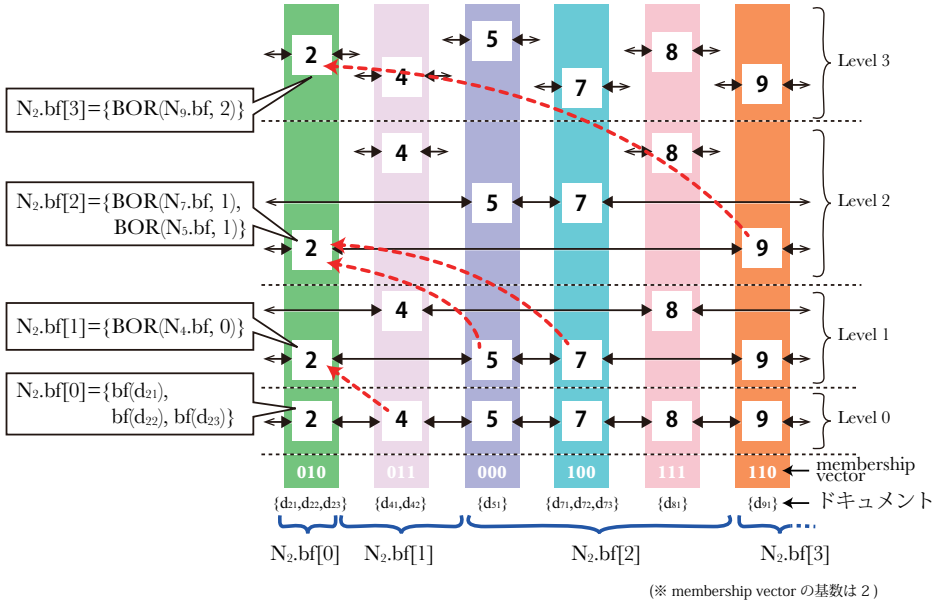

図 2 BF Skip graph の例

3.2 ノードの参加と離脱

BF Skip graph でノード n が参加する場合，まず n が保持するドキュメント d_i に対応する Bloom filter $b_i = \mathrm{bf}(d_i)$ を求め，$n.\mathrm{bf}[0]$ に格納する．なお，$b_i.ptr = n, b_i.doc = d_i$ としておく．次に，Skip graph と同じ手順を用いて BF Skip graph に挿入する（このためには，ノードは BF Skip graph に参加している任意のノードを知っている必要がある）．

Skip graph への挿入が完了したら，n は $n.\mathrm{bf}[l](l > 0)$ を初期化する．この処理の詳細は

Vol.2011-DPS-146 No.27
Vol.2011-CSEC-52 No.27
2011/3/11

```
1 bf_key = bf(keyword_1) ∨_bit bf(keyword_2) ∨_bit ...
2 send ⟨searchOp, bf_key, n, n.maxlevel⟩ to n
3 foundOP メッセージが送られてくるのを一定時間待つ.
```

図 3 検索を行うノード n の処理

3.4 節で述べる.

BF Skip graph からのノードの離脱処理は，Skip graph でのノードの離脱処理と同じである.

BF Skip graph では，ノードの参加にかかる時間は Skip graph と同様 $O(w \log_w N)$（N はノード数）であり，ノードが保持しているドキュメント数には依存しない.

3.3 検索アルゴリズム

ここでは BF Skip graph における検索アルゴリズムを述べる．$\vee_{\text{bit}}$，$\wedge_{\text{bit}}$ をそれぞれビット単位 OR，ビット単位 AND を表す演算子とする．検索するノードを n，検索キーワードを keyword_1，keyword_2, ... とする．各キーワードに対応する Bloom filter をビット単位 OR をとって集約した Bloom filter $(\text{bf}(\text{keyword}_1) \vee_{\text{bit}} \text{bf}(\text{keyword}_2) \vee_{\text{bit}} \ldots)$ を bf_{key} とする．指定したすべてのキーワードを含むドキュメントの Bloom filter b に対し，$b \wedge_{\text{bit}} bf_{\text{key}} = bf_{\text{key}}$ となることを利用して検索を行う.

n は，メッセージ $\langle \text{searchOp}, bf_{\text{key}}, n, n.\text{maxlevel} \rangle$ を n 自身に送信し，検索結果が含まれる foundOp メッセージの到着を一定時間待つ（**図 3**）.

メッセージ $\langle \text{searchOp}, bf_{\text{key}}, n, \text{level} \rangle$ を受信したノード p の動作を**図 4** に示す．p は，$p.bf[0 \ldots \text{level}]$ のすべての Bloom filter b に対して，指定したすべてのキーワードが含まれる可能性があるかどうかを $b \wedge_{\text{bit}} bf_{\text{key}} = bf_{\text{key}}$ が成立するか否かでチェックする．成立しない場合，b に対応する区間にはマッチするドキュメントは存在しない．成立する場合，マッチするドキュメントが存在する可能性があるため，b の取得元ノード $b.ptr$ に searchOp メッセージを転送する．なお，level = 0 で条件が成立した場合は，対応するドキュメントを foundOp メッセージで n に送信する.

なお，foundOp メッセージで n に通知されるドキュメントには，Bloom filter の偽陽性により誤った結果が含まれている可能性がある．この問題に対しては，n が当該ドキュメントに実際の検索キーワードが含まれるかどうかをチェックする，あるいは searchOp メッセージに実際の検索キーワードを入れておき，foundOp メッセージを送信する際にチェックするといった方法が考えられる.

```
1  when p receives ⟨searchOp, bf_key, n, level⟩ {
2    for (; level ≥ 0; level−−) {
3      foreach b ∈ n.bf[level] {
4        // bにbf_keyが含まれているかどうかをチェック
5        if (b ∧_bit bf_key = bf_key) {
6          if (level > 0) {
7            // レベルが 0以上の場合,bの取得元ノードへ searchOp メッセージを転送
8            send ⟨searchOp, bf_key, n, level−1⟩ to b.ptr
9          } else {
10           // レベルが 0の場合,該当するドキュメントをnへ送信
11           send ⟨foundOp, b.doc⟩ to n
12         }
13       }
14     }
15   }
16 }
```

図 4 searchOp メッセージを受信したノード p の処理

図 5 の N_2 がキーワード (keyword1, keyword2) を AND 検索する場合を例に説明する．N_2 が保持する Bloom filter は図中に示した．ここでは，N_4 が保持するドキュメント d_{41} と N_8 が保持する d_{81} に両方の検索キーワードが含まれているものとする．また，説明を単純にするため Bloom filter の偽陽性は発生しないものとする.

(1) N_2 はキーワード keyword1 と keyword2 から bf_{key} を求め，メッセージ $\langle \text{searchOp}, bf_{\text{key}}, N_2, N_2.\text{maxlevel}(=3) \rangle$ を N_2 へ送信し，foundOp メッセージの到着を待つ.

(2) searchOp メッセージを受信した N_2 は，$N_2.\text{bf}[0 \ldots 3]$ に含まれるすべての Bloom filter と bf_{key} とのビット単位 AND を計算し，bf_{key} のすべての “1” のビットが立っている Bloom filter を検索する．ここで，(a)N_7 から得た Bloom filter $(\text{BOR}(N_7, 1))$ と，(b) N_4 から得た Bloom filter $(\text{BOR}(N_4, 0))$ が条件を満たす．なぜなら，(a) の $\text{BOR}(N_7, 1)$ は $[N_7, N_9)_0$ の範囲のドキュメントに対応した Bloom filter であるため，d_{81} の Bloom filter を含み，また，(b) の $\text{BOR}(N_4, 0)$ は d_{41} の Bloom filter を含むためである．このため，N_2 はメッセージ $\langle \text{searchOp}, bf_{\text{key}}, N_2, 1 \rangle$ を N_7 に，また，$\langle \text{searchOp}, bf_{\text{key}}, N_2, 0 \rangle$ を N_4 に送信する.

Vol.2011-DPS-146 No.27
Vol.2011-CSEC-52 No.27
2011/3/11

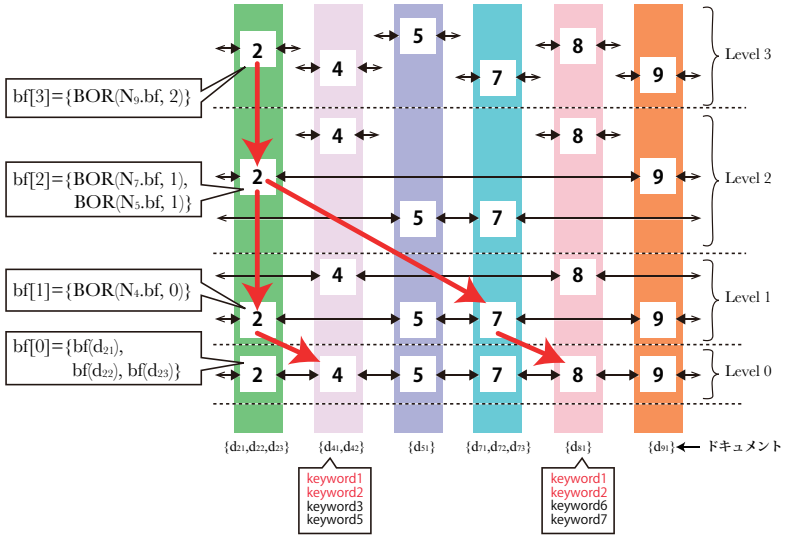

図 5 searchOp メッセージの流れ

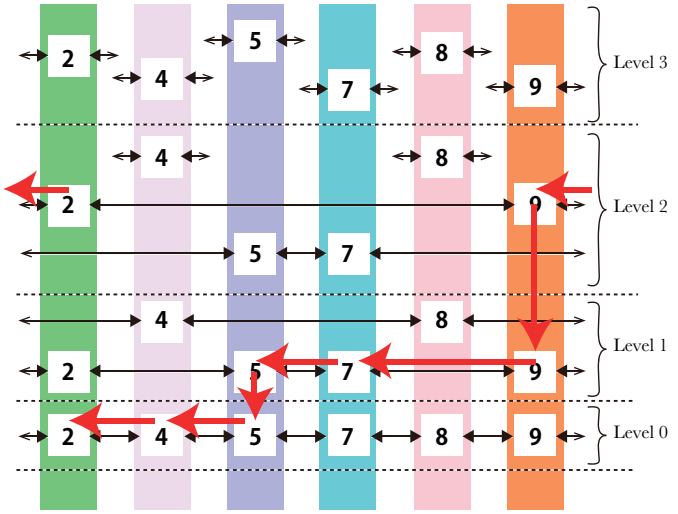

図 6 N_2 の Bloom filter 更新時の updateOp メッセージの流れ

(3) searchOp メッセージを受信した N_7 は，同様に N_7.bf[0...1] に含まれるすべての Bloom filter と bf_{key} とのビット単位 AND を計算する．ここで，N_7 が保持する BOR(N_8, 0) が条件を満たすため，⟨searchOp, bf_{key}, N_2, 0⟩ を N_8 に送信する．

(4) searchOp メッセージを受信した N_4 と N_8 は，同様に N_4(or N_8).bf[0] の Bloom filter と bf_{key} とのビット単位 AND を計算する．それぞれ d_{41} と d_{81} が条件を満たすため，メッセージ ⟨foundOp, d_{41}⟩ と ⟨foundOp, d_{81}⟩ を N_2 に送信する．

BF Skip graph での検索時間は Skip graph の高さ (maxlevel) に比例するため，$O(\log_w N)$ (N はノード数) となる．検索に必要なメッセージ数はマッチするドキュメント数によって変化する（4 章でシミュレーションにより評価する）．

3.4 各ノードの Bloom filter 更新アルゴリズム

ノード n が保持する Bloom filter (n.bf) を更新するアルゴリズムを述べる．

n.bf[0] は，n 自身が保持するドキュメントに対応する Bloom filter であり，n が BF Skip graph に参加した時，および n のドキュメントに変更があった時に更新する．以下では，n.bf のレベル 1 以上の部分を更新（初期化）する方法を述べる．ノードは参加・離脱する可能性があり，また他のノードが保持するドキュメントに変更がある場合もあるため，この処理は定期的に実行する．

BF Skip graph での Bloom filter の更新の流れは，集約 Skip graph での集約値更新処理と同様である．n は，各レベルの Bloom filter (n.bf) を収集するために，updateOp メッセージをレベル maxlevel − 1 の左ノード (n.left[maxlevel − 1]) に送信する．updateOp メッセージは左ノードに転送されるが，左ノードが updateOp メッセージの発行元ノードの場合はレベルを 1 つ下げ，最終的にレベル 0 で発行元ノードに戻る．**図 6** は N_2 が送信した updateOp メッセージの流れを示している．

updateOp メッセージは Bloom filter の集合の配列 bf_{update} を保持する．メッセージがレベル l で転送される間，通過するノードの BOR($bf[l], l$) を $bf_{\text{update}}[l+1]$ に追加する．メッセージが発行元ノード n に戻ると，bf_{update} によって n.bf のレベル 1 以上の部分を更新する．アルゴリズムを**図 7**，**図 8** に示す．

updateOp メッセージは，各レベルで左ノードに $O(w)$ 回転送されるため，updateOp メッセージが発行元ノードに戻るまでには $(w \log_w N)$ ホップ（N はノード数）必要である．

各ノードは Bloom filter を定期的に更新する．ノード n がレベル l で保持する Bloom filter 集合 n.bf[l] の正確性は，$(n, n.\text{right}[l])_{l-1}$ に含まれるノードのレベル $l-1$ の Bloom

Vol.2011-DPS-146 No.27
Vol.2011-CSEC-52 No.27
2011/3/11

```
1  bf_update: Bloom filter の集合の配列．初期値は∅
2  // レベルmaxlevel−1の左ノードへupdateOp メッセージを送信
3  send ⟨updateOp, n, bf_update, maxlevel−1⟩ to n.left[maxlevel−1]
```

図 7 Bloom filter を更新するノード n の処理

filter の正確性に依存する．このため，あるノードの挿入や離脱，レベル 0 での Bloom filter の変更が全ノードに伝搬されるのは，各ノードが Skip graph の高さ ($O(\log_w N)$) の回数だけ更新処理を実行した後になる．つまり，あるノードのドキュメントが検索可能になるには，そのノードが参加してから $O(\log_w N)$ 時間経過した後である．

```
1  when p receives ⟨updateOp, n, bf_update, level⟩ {
2    if (n = p) {
3      // メッセージが発行元ノードnに戻ってきた場合
4      p.bf[1 . . . maxlevel] ← bf_update[1 . . . maxlevel]
5    } else {
6      b ← BOR(p.bf, level)
7      b.ptr ← p
8      bf_update[level+1] ← bf_update[level+1] ∪ b
9      // 左ノードが発行元ノードならばレベルを下げる
10     while (left[level] = n ∧ level > 0) {
11       level ← level−1
12     }
13     // 左ノードに転送
14     send ⟨updateOp, n, bf_update, level⟩ to p.left[level]
15   }
16 }
```

図 8 updateOp メッセージを受信したノード p の処理

図 6 で updateOp メッセージが N_2 に戻ってきたとき，$bf_{\text{update}}[3] = \{\text{BOR}(N_9, 2)\}$, $bf_{\text{update}}[2] = \{\text{BOR}(N_7, 1), \text{BOR}(N_5, 1)\}$, $bf_{\text{update}}[1] = \{\text{BOR}(N_4, 0)\}$ であり，N_2 はこれを用いて N_2.bf のレベル 1 以上の部分を更新する．

4. 評　　価

BF Skip graph をシミュレーションにより評価した．

ドキュメントとして，基本英単語 5,800 語からランダムに 300 語を抜き出したテキストファイルを用意した．同様のドキュメントを 100 個用意し，各ノードはこの 100 個の中からランダムに選ばれた複数個のドキュメント (最小で 1 個，最大で 5 個) を保持する．Bloom Filter のビット長は 10240 とし，4 個のハッシュ関数を使用した．

まず，指定したキーワードにマッチするドキュメントがネットワーク中に 1 つしかない場合の検索にかかる時間を**図 9** に示す．横軸はノード数，縦軸は検索にかかるホップ数である．なお，検索にかかるホップ数は，検索処理を開始してから，検索要求にマッチするドキュメントを保持するすべてのノードに searchOp メッセージが到達するまでのメッセージ転送の数としている．試行は 10 回行い，平均値をプロットした（以下同様）．検索ホップ数はノード数の log オーダであり，また membership vector の基数を大きくするとホップ数が減少することがわかる．

次に，指定したキーワードにマッチするドキュメントの数による傾向を調べるため，ノード数を 500 に固定し，マッチするドキュメントが全ノードの r% に存在する場合の検索にかかるホップ数を測定した（r は 1%から 100%まで変化させた）．結果を**図 10** に示す．マッチするドキュメントが増加してもホップ数はほとんど増加しない（定数オーダ）ことが確認できる．

また，このときの検索で送信した searchOp メッセージの総数も測定した（**図 11**）．指定したキーワードにマッチするドキュメントを保持するノードが k 個存在する場合，少なくとも k ノードに searchOp メッセージを送信する必要がある．searchOp メッセージ数が k に比べてどの程度大きいのかのオーバーヘッドをみるため，縦軸は検索に要したメッセージ数を k で割った値としている．k が大きくなるほどオーバヘッドが少なくなっている．これは k が大きいほど searchOp メッセージを受信したノードがマッチするドキュメントを持っている確率が上がるためである．

最後に，Bloom filter を更新するための updateOp メッセージが発行元ノードに戻るために必要なホップ数を**図 12** に示す．基数が大きいほどホップ数が増えているが，これは updateOp メッセージを横方向に転送する回数が増えるためである．

Vol.2011-DPS-146 No.27
Vol.2011-CSEC-52 No.27
2011/3/11

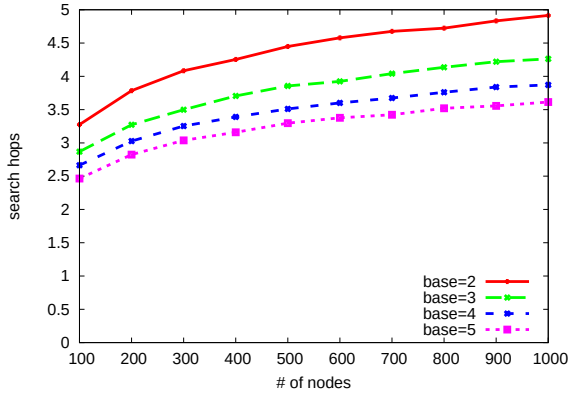

図 9 単一ドキュメントにマッチする場合の検索ホップ数

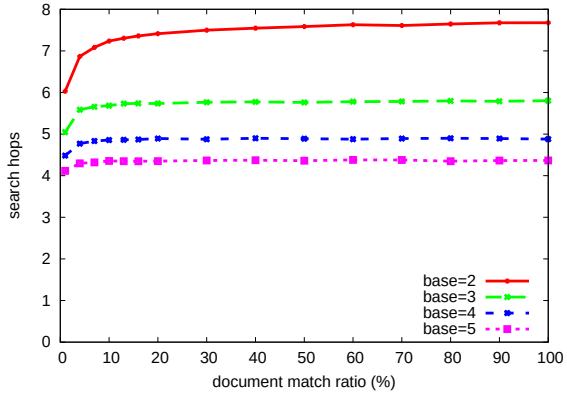

図 10 マッチするドキュメントの割合を変化させたときの検索ホップ数

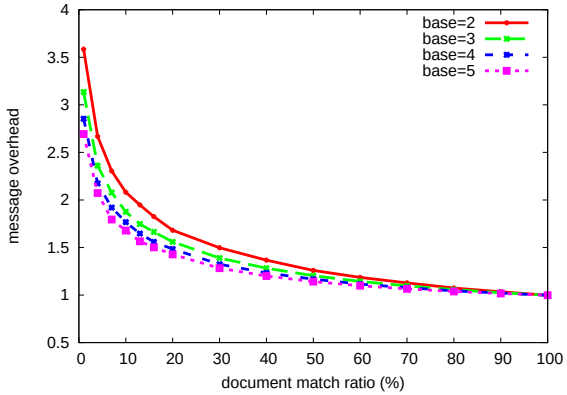

図 11 マッチするドキュメントの割合を変化させたときの検索メッセージのオーバヘッド

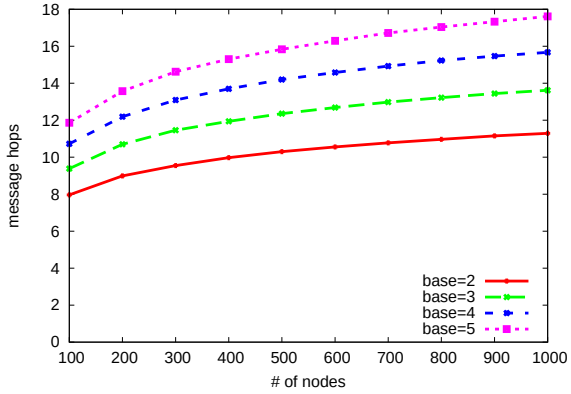

図 12 updateOp メッセージのホップ数

5. おわりに

本稿では，ネットワーク上に分散した多数のドキュメントに対して複数キーワードによる AND 検索を効率よく行うための手法 (BF Skip graph) について述べた．BF Skip graph では検索キーワードの数やドキュメントの数に依存せず，AND 検索を $O(\log_w n)$ 時間 (n はノード数，w は membership vector の基数）で実行できる．また，必要なメッセージ数のオーバーヘッドも少ない．

本研究の今後の課題をいくつか挙げる．

- 提案手法では AND 検索のみを扱っているが，AND と OR を組み合わせた条件で検索ができるように拡張する．

Vol.2011-DPS-146 No.27
Vol.2011-CSEC-52 No.27
2011/3/11

- 提案手法では大量のドキュメントがマッチした場合，検索するノードへ大量のデータが送られることになる．このため，検索結果の上限数を決められるような方法を検討する．

参 考 文 献

1) Bloom, B.H.: Space/Time Trade-offs in Hash Coding with Allowable Errors, *Communications of the ACM*, Vol.13, No.7, pp.422–426 (1970).
2) Abe, K., Abe, T., Ueda, T., Ishibashi, H. and Matsuura, T.: Aggregation Skip Graph: A Skip Graph Extension for Efficient Aggregation Query, *Proceeding of the 2nd International Conference on Advances in P2P Systems (AP2PS 2010)*, pp. 93–99 (2010).
3) Aspnes, J. and Shah, G.: Skip Graphs, *ACM Transactions on Algorithms*, Vol.3, No.4, pp.37:1–37:25 (2007).