LL_004

P2Pネットワーク上の文書検索手法の研究
The full text retrieval search system on P2P network.

成毛 源樹† Motoki Naruke
山口 和紀‡ Kazunori Yamaguchi

1. はじめに

P2Pネットワークの利点を生かし、様々なシステムを構築することを考えると、サーバを使わずに、効率的かつ高機能な検索機構を実現することが必要である。例えばWebサイトのhtmlファイルを保持しているノード群によって形成されたP2Pネットワークで、それらファイルを対象とした全文検索システムが実用化されることで、Web検索エンジンが実現される。

そこで本研究は、P2Pネットワーク上のノードが保持している全ての文書に対して、効率的かつスケーラブルに全文検索を行えるシステムを実現する手法を提案する。P2Pネットワークを形成するノードの数、各ノードが保持している文書の数は時間の経過とともに増加することを前提としているので、スケーラブルであることが重要である。また、システムの実現にあたり、"検索語への適合度が高い文書を検索できること"を目的とする。検索語に対する文書の適合度の評価基準としては様々なものが提案されているが、本研究ではスコア値としてtf-idfを用いて、"検索語のスコア値が高い文書＝検索語への適合度が高い文書"とする。

2. 既存の探索手法による実現とその問題点

P2Pにおける探索手法には、現在のところGnutella [1]で用いられた、バケツリレーのようなFlooding方式と、Chord [3]をはじめとする、ハッシュ関数により決められた空間にノードとコンテンツをマッピングすることで探索を効率化するDHT(Distributed Hash Table)方式がある。以下、これらの探索手法を用いて、目的とする全文検索システムを実現できるか考察する。

2.1 Flooding方式を用いた実現手法

Flooding方式を用いて、P2Pネットワーク上のノードが保持している全ての文書から、検索語wのスコアが高い文書を検索するアルゴリズムを、検索を始めるノードを$node_s(IP = ip_s)$として以下に示す。

1. $node_s$は、隣接ノードにクエリ$= (TTL,\ ip_s,\ w)$を送信する。
2. クエリを受信したノードは、自分が保持している文書を対象としてwで全文検索を行い、$node_s$に検索結果を送信する。
   $TTL = TTL - 1$として、$TTL > 0$なら隣接ノードにクエリを送信する。

TTL（Time To Live）とは、"クエリを送信する残り回数"を表し、TTLが0になるまでクエリの送信は続く。

Flooding方式を用いた手法では、ネットワーク全体にクエリを送信する必要がある。1ノードに隣接するノードの数をn、TTLをtとすると、$n^t \geq$ ノード数 を満たすようにTTLを選ばなければならず、n^t回の通信が必要となる。また、1クエリの大きさをqバイトとすると、クエリをネットワーク全体に送信するときに生じる通信量はqn^tバイト となる。これら通信量/回数はノード数の増加に伴い、爆発的に増加してしまう[2]。よって、この手法はスケーラブルではないことがわかる。

2.2 DHTを用いた実現手法

DHTを用いた全文検索では、一般的な全文検索システムで用いられる転置インデックスをDHT上に構築することで全文検索を実現する方法が提案されている[4]。ネットワーク全体の文書に対して一貫したインデックスを構築するので、検索結果の取捨選択も可能である。

一般的な転置インデックスはkey(索引語)に対して、{(文書名, 出現位置)}といった情報を登録したものである。keyとしては単語や2gramが使われることが多い。[4]では、どのような転置インデックスを用いているのか明記されていないので、本節ではkeyとして2gramを用いて、$ID = hash(key)$のノード$node_{key}$に$index_{key} = \{(IP,$ 文書名, 出現位置, 文書サイズ$)\}$を構築した場合について考察する。

keyとして2gramを用いる理由は、特定の文書形式に特化したシステムではなく、様々な文書形式に柔軟に対応できるようにするため、検索漏れをなくすためである。

DHT上に2gramをkeyとした転置インデックスが構築されているとき、P2Pネットワーク上のノードが保持している全ての文書から、検索語wのスコア上位n文書を検索するアルゴリズムを、検索を始めるノードを$node_s(IP = ip_s)$として以下に示す。

1. $node_s$は検索語$w = s_1s_2...s_m$を構成する全ての2gram$= \{s_1s_2,\ s_2s_3,\ ...,\ s_{m-1}s_m\}$より
   $1 \leq i \leq m-1$に関して$bg_i = s_is_{i+1}$として
   $list_{bg} = \{(i,\ bg_i)\}$を構築し、
   $size(index_{bg_i})$が小さい順にソートする。
2. $node_s$は空の$index_w$を用意する。
   $list_{bg}$の先頭からi, bg_iを取り出し、$node_{bg_i}$に$(index_w,\ list_{bg},\ ip_s)$を送信する。
3. $node_{bg_i}$は$list_{bg}$の先頭からi, bg_iを取り出す。
   $index_w$が空なら$index_{bg_i}$を$index_w$とする。
   空ではないなら$index_w$と$index_{bg_i}$の要素のうちIP、文書名が等しく、出現位置がiずれているものだけを集めて$index_w$を再構築する。
   $index_w$が空なら$node_s$に$index_w$を送信する。
   空ではないなら$list_{bg}$の先頭を削除する。
   $list_{bg}$が空なら$index_w$を$node_s$に送信する。
   空ではないなら$list_{bg}$の先頭からi, bg_iを取り出し、

†東京大学大学院 総合文化研究科 広域システム科学系
Graduate School of Arts and Sciences,
The University of Tokyo. motoki@graco.c.u-tokyo.ac.jp
‡東京大学 情報基盤センター
Information Technology Center, The University of Tokyo.

$node_{bg_i}$ に $(index_w, list_{bg}, ip_s)$ を送信する。

4. $node_s$ は $index_w$ が空なら検索を終了する。
   空ではないなら、要素の数、文書サイズから w に関する文書のスコアを算出し、スコア上位 n 文書を取得する。

以上により構築された $index_w$ を用いることで検索語 w がどのノードの、どの文書の、どの位置に出現するかがわかる。また、w に関する文書のスコアも算出することが可能なので、ネットワーク全体からスコアが高い文書を取得することが可能となる。

ノード数が N のとき、DHT 上で $node_{bg_i}$ 探索時に発生する平均通信回数は $\log_2 N$ 回である。log スケールであるため、ノード数が増加しても通信回数が爆発的に増加することはない。しかしインデックス構築時のクエリが各 $index_{bg_i}$ の共通部分を含むため、通信量が大きくなってしまう可能性がある。

ある key が文書中に出現する平均回数を f、各ノードが持つ文書の数を D、key に関してインデックスに登録される情報を m バイトとした場合、$index_{key}$ の大きさは $NfDm$ バイトとなり、インデックスのサイズはノード数、文書数、出現回数に対して線形に増加することがわかる。

例えばノード数 10000、$f = 10$、各ノードが持つ文書数 10000、$m = 100$ バイトとすると、1 GB となってしまう。いくら通信回数が少なくても、これだけの大きさのデータを送受信することは現実的ではない。

つまり、この手法は通信回数に関してはスケーラブルだが、検索に用いるインデックスのサイズが大きくなってしまうと、スケーラブルではなくなってしまう。インデックスのサイズが大きくなったときに、[4] では Bloom Filters [5] によるデータの圧縮が考察されているが、[5] によると Bloom Filters による圧縮は最大でも数十分の一程度の圧縮率なので本質的な解決とはなっていない。

3. スコアに着目したインデックスサイズ削減手法の提案

インデックスのサイズが大きくなってしまうのは、文書毎、出現位置毎の情報をインデックスに登録しているためである。そこで本研究では「スコアを用いて結果を選出する」ことに着目し、ノード毎に、"key を含む文書のスコアの最高値" を登録した「S1index」を提案する。

本章では、S1index の詳細と、S1index を用いて、P2P ネットワーク上のノードが保持している全ての文書から、検索語 w のスコアが高い文書を検索できることを検証する。

3.1 S1index

S1index は $ID = hash(key)$ のノード $node_{key}$ に、$index_{key} = \{(ip_i, max_{(i,key)})\}$ を構築したものである。ここで ip_i = あるノード $node_i$ の IP、$max_{(i,key)}$ = $node_i$ 中で key を含む文書のスコアの最高値 である。

S1index ではサイズの増加がノード数に対してのみ線形となり、手法 2.2 で生じた、インデックスのサイズの問題が解決される。

例えばスコアを 10 バイト で表し、ノード数が 10000 のとき、各ノードが 10000 文書を持っていたとしても、インデックスのサイズは高々 $100KB$ である。

本手法では文書のスコア値として tf-idf を用いるが、ネットワーク全体で一貫したスコア付けを行うために、ネットワーク中のノードは $node_{key}$ に、スコア値ではなく文書中の key の出現頻度と、自分が保持している key を含む文書数、自分が保持している文書の総数を登録する。こうしておくことで $node_{key}$ は、ネットワーク上の総文書、ネットワーク上で key を含む文書の総数を把握でき、tf-idf の定義式 (1) を用いて、必要なときにスコア値を算出することが出来る。

$$\text{スコア値} = \text{出現頻度} \cdot \log\left(\frac{\text{ネットワーク上の総文書数}}{key\text{を含む文書数}}\right) \quad (1)$$

key を含む文書数、ネットワーク上の総文書数は時間の経過とともに変化するので、変化するたびに更新する必要がある。しかし頻繁に更新してしまうとトラフィックの問題を引き起こしかねないので、ネットワーク上の各ノードは、自分の max_{key} が変化したときに、それぞれを更新する。

3.2 2gram のスコアと単語のスコアの相関の検証

S1index を用いて、P2P ネットワーク上のノードが保持している全ての文書から、検索語 w のスコアが高い文書を検索できることを示すためには、次の仮定

「ある単語を構成する 2gram のスコアが高い文書は、
単語のスコアも高い文書である」

が成り立てばよい。これはつまり、'*do*' のスコアが高い文書と、'*og*' のスコアが高い文書は、単語 '*dog*' のスコアが高い文書である、という仮定である。

この仮定を検証するために、データセットとして http://www.gutenberg.org/ より取得した文書 (総文書数: 10598、総容量: 4136MB) を用いて、21217 単語について実験を行った。

単語セットの平均文字数は 9.58 で、文字数の分布は図 1 のようになった。横軸は文字数で、縦軸は文字数が占める全体の割合である。

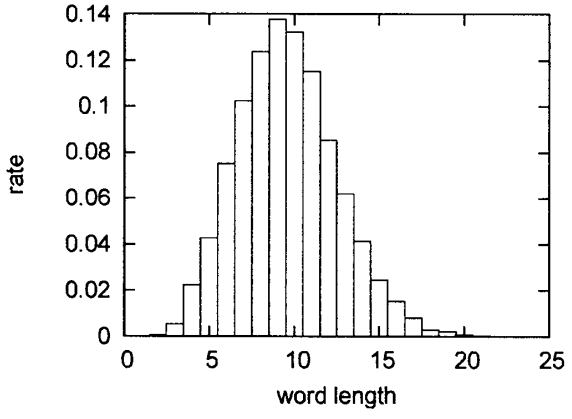

図 1: 使用単語セットの文字数の分布

3.2.1 2gram のスコアと単語のスコアの相関

ある単語を構成する 2gram のスコアの平均と、単語のスコアの相関係数を算出した。単語を構成する 2gram 全てを含んでいても、実際には単語を含まない文書は除いた。図 2 は相関係数の分布を表す図で、横軸は相関係数、縦軸は全体に占める割合となっている。

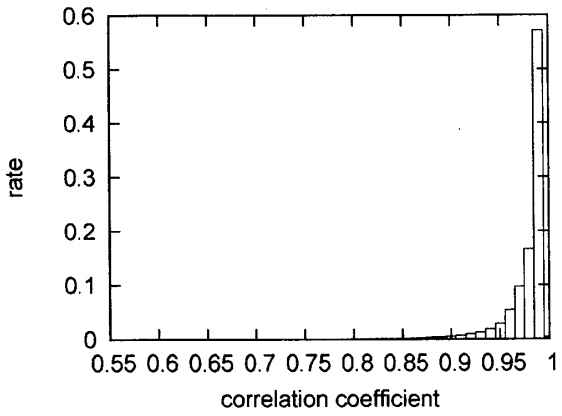

図 2: 2gram のスコアと単語のスコアの相関係数の分布

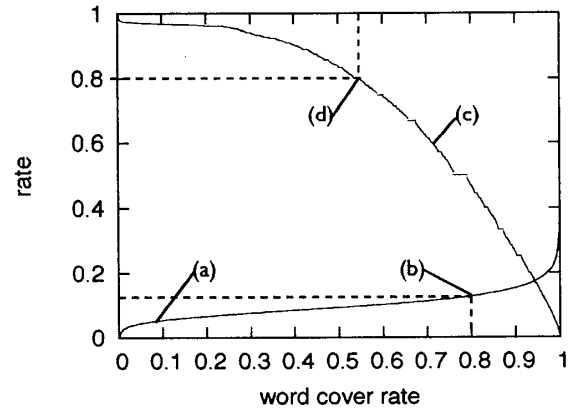

図 3: スコア上位文書を選出

全単語について相関係数を算出した結果、平均は 0.983 となった。単語の中には相関係数が -1 となるものもあったが、これは出現頻度が非常に低い単語で、その次に相関係数が低い単語で 0.338、以降は 0.553 以上となった。

以上の結果から単語を構成する 2gram のスコアの平均と、単語のスコアは、総じて高い相関を示すと言える。

3.2.2 2gram のスコアによる単語スコア上位文書の選出

3.2.1 により、2gram のスコアと単語のスコアに高い相関が認められたので、2gram のスコアを用いることで単語のスコアが高い文書をどれだけ効率的に選出できるか確かめた。

単語のスコアが上位 5%以内であるものを、単語のスコアが高い文書と考え、「スコア上位文書」と呼ぶ。以下ではある単語のスコア上位文書の 80%を選出するために、単語を構成する 2gram 全てを含む文書から、上位何%のものまでを選出すればよいかを調べた。また、選出された文書が実際に単語を含む割合も調べた。

図 3 の (a) はスコア上位文書を選出するために、単語を構成する 2gram 全てを含む文書からスコア上位何%のものまでを選出すればよいかを示すグラフである。横軸は実験に使用した 21217 単語のうち、条件に従う単語の割合である。例えば点 (b) を見ると、2gram のスコアの平均が上位 10%となるような文書を選出することで、80%程度の単語でスコア上位文書を選出することができる、ということがわかる。

図 3 の (c) は 2gram のスコアの平均が (a) の基準を満たすような文書を選出したときに、それらの文書が実際に単語を含んでいる割合を示すグラフである。横軸は単語のカバー率である。例えば点 (d) を見ると、(a) に従って 2gram の平均スコア上位の文書を選出することで、選出した文書の 80%以上に実際に含まれるような単語が 50%以上あることがわかる。

以上の実験により、2gram の平均スコアが高い文書を選出することで、単語のスコアが高い文書を選出できることがわかった。これにより、S1index に登録されている 2gram のスコアを用いて、P2P ネットワーク上のノードが保持している全ての文書から、検索語 w のスコアが高い文書を持つノードを選出することが可能であることがわかる。

4. S1index を用いた検索

3 章では S1index により、インデックスのサイズの問題が解決できることと、S1index に登録された 2gram のスコアを用いることで、単語のスコア上位文書を検索できることを検証した。

本章では S1index を用いた検索アルゴリズムを示し、発生する通信量を手法 2.1、2.2 と比較する。

4.1 S1index による検索

DHT 上に 2gram を key とした S1index が構築されているとき、P2P ネットワーク上のノードが保持している全ての文書から、検索語 w のスコア上位 n 文書を検索するアルゴリズムを、検索を始めるノードを $node_s(IP = ip_s)$ として以下に示す。

1. $node_s$ は検索語 $w = s_1s_2...s_m$ を構成する全ての 2gram $= \{s_1s_2,\ s_2s_3,\ ...,\ s_{m-1}s_m\}$ より $1 \leq i \leq m-1$ に関して $bg_i = s_is_{i+1}$ として $list_{bg} = \{(i,\ bg_i)\}$ を構築し、$size(index_{bg_i})$ が小さい順にソートする。
2. $node_s$ は空の $index_w$ を用意する。$list_{bg}$ の先頭から i, bg_i を取り出し、$node_{bg_i}$ に $(index_w,\ list_{bg},\ ip_s)$ を送信する。
3. $node_{bg_i}$ は $list_{bg}$ の先頭から i, bg_i を取り出す。$index_w$ が空なら $index_{bg_i}$ を $index_w$ とする。空ではないなら $index_w$ と $index_{bg_i}$ の要素のうち

IP が等しいものだけを集め、スコアを足し合わせることで $index_w$ を再構築する。
$index_w$ が空なら $node_s$ に $index_w$ を送信する。
空ではないなら $list_{bg}$ の先頭を削除する。
$list_{bg}$ が空なら $index_w$ を $node_s$ に送信する。
空ではないなら $list_{bg}$ の先頭から i, bg_i を取り出し、$node_{bg_i}$ に $(index_w, list_{bg}, ip_s)$ を送信する。

4. $node_s$ は $index_w$ が空なら検索を終了する。
空ではないなら $index_w$ をスコアが高い順にソートし、先頭から順に結果 $= \{(w$ 出現頻度, IP, 文書名$)\}$ の数が n 以上になるまで、$(w, worst)$ を送信し、各ノードの検索結果を取得する。$worst$ の初期値は $indx_w$ の上から n 番目のスコアとする。
このとき検索結果と一緒に各ノードにおける w 出現文書数も送信してもらう。
5. $(w, worst)$ を受信したノードは自分が保持している文書を対象として w で全文検索を行い、検索結果のうちスコア $worst$ 以上の上位 n 文書を用いて、結果 $= \{(w$ 出現頻度, IP, 文書名$)\}$ を構築し、w 出現文書数と一緒に $node_s$ に送信する。

S1index を用いた検索では、通信回数は手法 2.2 と同じだが、インデックスのサイズが小さいので、$index_w$ を構築する際の通信量の削減が期待される。

こうして構築された $index_w$ からは "w に関してスコアが高い文書を持つ可能性が高いノード" しかわからないので、ネットワーク全体からスコア上位 n 文書を取得するためには、ノードそれぞれにクエリを送信し、ノード毎の検索結果を集め、その中からスコアが高い文書を選出する必要がある。このとき、結果として返されるスコアはノードにローカルなものであるため、他のノードの結果と比較することができない。そこで各ノードの文書中の出現頻度を仮のスコアとして扱い、同時に w 出現文書数を累積していき、必要な時にスコアを算出できるようにする。この方法は単語一つを検索するときには不要であるが、二つ以上の単語それぞれの検索結果に重みを付けるときに必要となる。

4.2 通信量比較

ノード数を N、S1index の各要素を d バイトで表すと $index_w$ 構築時に発生する通信量は最悪の場合で $(m-1)dN$ バイト である。最悪の場合というのはネットワーク上の全てのノードが、検索語 w を構成する 2gram 全てを含む文書を持っている場合である。

各ノードに検索クエリを送信するとき、クエリを q バイト とすると最悪で qN バイト の通信量が発生する。最悪の場合というのはネットワーク上の全てのノードが w を含む文書を一つも持っていない場合である。

既存手法を用いて全文検索を実現した 2.1、2.2 との通信量を比較すると表 1 のようになる。それぞれの変数は以下の通りである。
$N =$ ノード数, $D =$ 文書数, $f =$ 出現回数,
$m =$ 検索語の文字数 $- 1$, $q =$ クエリのサイズ,
$d =$ スコア表現サイズ, $n =$ 結果として欲しい文書数

表 1: 各手法で発生する通信量の比較

	手法 2.1	手法 2.2	S1index
インデックス探索	—	$log_2 N$	$log_2 N$
インデックス取得	—	$mfDN$	mdN
クエリ伝搬	qN	qn	qN

5. まとめ

本研究では「スコアを用いて結果を選出する」ことに着目し、DHT 上に 2gram を key として、ノード毎に、"key を含む文書のスコアの最高値" を登録した「S1index」を提案した。S1index のサイズはノード数に対して線形で、一般的な転置インデックスと比較して、登録、取得時の通信量を大幅に削減できることを示した。また、「ある単語を構成する 2gram のスコアが高い文書は、単語のスコアも高い文書である」という仮定を検証し、S1index に登録されている 2gram のスコアを用いて、ネットワーク上の全ての文書から、検索語 w のスコアが高い文書を検索できることを示した。

しかし S1index は "単語 w に関してスコアが高い文書を持つ可能性が高いノード" しかわからないので、ネットワーク上の全てのノードが単語 w を含む文書を持っていない場合、Flooding を用いた手法と同等の通信量が発生してしまうという問題点がある。

今後は実際の大規模な文書データを用いて実験を行い、前述の問題点の検証および通信量/回数、検索精度などを、計測/評価することで本手法の有効性を確かめる。また、ネットワーク上の各ノードの検索頻度、インデックス更新頻度など、実験による計測/評価が難しい要素に関しては、数学的モデルを作成し、シミュレーションを行うことで振る舞いを明らかにしていきたい。

参考文献

[1] Gnutella, http://www.gnutella.org/

[2] Jordan Ritter,
"Why Gnutella Can't Scale. No, Really",
http://www.darkridge.com/~jpr5/doc/gnutella.html

[3] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", ACM SIGCOMM 2001.

[4] Patrick Reynolds and Amin Vahdat,
"Efficient Peer-to-Peer Keyword Searching",
ACM/IFIP/USENIX International Middleware Conference 2003.

[5] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors", Communications of the ACM, 1970.