協調サーチエンジンにおける継続検索のための先読みキャッシュ方式

佐藤永欣† 山本 崇† 西田 喜裕† 上原 稔† 森 秀樹†

†東洋大学工学部情報工学科

サーチエンジンを構築する場合、集中型サーチエンジンでは大量の文書の収集やインデックスの管理が問題になる。そこで、分散したサーチエンジンが協調してサーチを行う協調サーチエンジン (Cooperative Search Engine, CSE) を提案した。あるサーチエンジンに情報が無くても、情報を持っている他のサーチエンジンを教えてもらうことで間接的に検索できる。本稿では検索時における先読みキャッシュの管理について述べる。

Look Ahead Cache for Next 10 in Cooperative Search Engine

Nobuyoshi Sato†, Takashi Yamamoto†, Yoshihito Nishida† Minoru Uehara†, Hideki Mori†

†Department of Information and Computer Sciences, Toyo University

A centralized search engine has a few problems on collecting a large number of documents and managing index information. So, we proposed Cooperative Search Engine(CSE) in which more than one distributed search engines cooperate. Even if a search engine does not have index information, it can get the information by asking other search engines which have the information. In this paper, we describe managements of look ahead cache when searching.

1 はじめに

協調サーチエンジン (Cooperative Search Engine, CSE)[11][12] は、既存の集中型検索エンジンの問題点を解決すべく提案された、一部の専門知識を有する複数の局所メタサーチエンジン (Local Meta Search Engine, LMSE) が互いに協調することで検索を行う分散型協調サーチエンジンである。通常のサーチエンジンでは、あるドメイン内の全ての文書を対象とした検索を1台のマシンで行う。このため、サーチエンジンとなるマシンではドメイン内の全ての文書に関するデータベースを作成し、所有することになる。このデータベースをインデックスと呼ぶが、これを1台のマシンに所有させるのは、インデックス作成用の文書の収集、インデックスの保管、検索時の負荷の面からも大きな負担となる。協調サーチエンジンでは、小規模なメタサーチエンジンが協調して動作するため、これらの問題から開放される。

分散型のサーチエンジンは更新時間の短縮、インデックスの保管の分散化による保管コスト軽減など、集中型サーチエンジンと比較してイントラネットに適した利点を持つ。しかしその一方で、検索時に相互に通信するため検索に要する時間が長い、等の問題点がある。この問題に対処するため、検索結果をキャッシュし、2回目以降の検索を高速化するキャッシュサーバを協調サーチエンジンに導入した [14]。しかし、結果として無駄になるキャッシュデータが多い、キャッシュサーバの負荷が高いなどの問題点があり、本論文ではこれらの問題点を解決する方法を提案する。

本論文の構成は以下の通りである。第2節ではサーチエンジンにおける分散化と分散型サーチエンジンにおけるキャッシュについての関連研究、第3節ではCSEの概要と動作について述べる。第4節ではキャッシュアルゴリズムについて述べる。最後にまとめを述べる。

2 関連研究

サーチエンジンの構成は収集部分と検索部分の2つに分けることができる。このうち収集部分の分散化はJEIDA[3] 等で行なわれている。

このような分散型のロボットによる文書の収集と、集中型のサーチエンジンによる検索の組合せは、現在一般的に行われていると考えられる。この方式は文書の収集範囲が広い場合は有効であるが、イントラネッ

トのように文書を収集する範囲が狭く、極短時間の更新周期が要求される場合には、不十分である。

収集方法を工夫して更新時間を短縮しているのはFreshEye[8] 等である。しかし、これらは本質的に集中型サーチエンジンであり、更新時の負荷が問題である。FreshEye では 1 日前までの情報まで検索可能である。また、Infoseek[6] では各サイトがその日の更新分を robots1.txt として要約する方法で更新を効率化している。しかし、これらの方式ではイントラネットに要求される更新時間を実現するのは困難である。

検索部分の分散化は、検索をワークステーションクラスタで行う inktomi[7] や、インターネット上に分散している情報を自律的に統合する機能をもった情報検索システムの "Ingrid"[4] 等がある。Ingrid では、Ingrid サーバが Web サーバ内の情報のキーワードの索引付けと管理を自動的に行ない、他の Ingrid サーバと通信を行なって、類似した内容を持つ情報間でリンクをはりネットワークを形成し、検索できるようにしている。また、このネットワークの拡張は新たに Ingrid サーバを追加するだけですむという高い拡張性も備えている。しかし、運用に大きなリソースを要求するため導入コストが高い。

イントラネットやアドホックネットワークでの利用を考えた、モバイルエージェントを用いた分散型の検索システムもある [9][10]。これは Namazu[5] のインデクサとデータベースを用いるシステムである。Namazu はじめとする多くのサーチエンジンでは CSE も含めて $tf \cdot idf$ 法をスコアの計算に用いるが、$tf \cdot idf$ 法の計算に必要なデータを得るのが困難である [13]。CSE では検索時の通信回数を減らすことに主眼をおいてこの問題を解決しているが、このシステムでは更新作業の繁雑さを解消している。また、検索結果の「次の 10 件」を表示する際の効率の問題も解決している[1]。

3 CSE の構成と動作の概要

3.1 現在の CSE の構成と問題点

現在の CSE の構成と検索時の動作の概略を Fig.1 に示す。CSE の 3 つの構成要素の役割を以下に示す。

Local Search Engine(LSE) ある特定の Web サーバにある文書の検索を行うサーチエンジン。現在、Namazu と SGSE が使用可能である。

[1] [10] では継続検索と呼んでいる。

Local Meta Search Engine(LMSE) ユーザーとの対話及び、LSE の管理を行う。導入が容易なように Perl で記述された CGI として実装されている。

Location Server(LS) 与えられたキーワード、検索式について詳しい情報を持っている LMSE を探す。また、LSE のデータベースを更新する指示を LMSE に出すなど、CSE 全体を制御する。

Cache Server(CS) LMSE と LS の間で検索結果を検索式 E 毎にキャッシュする。

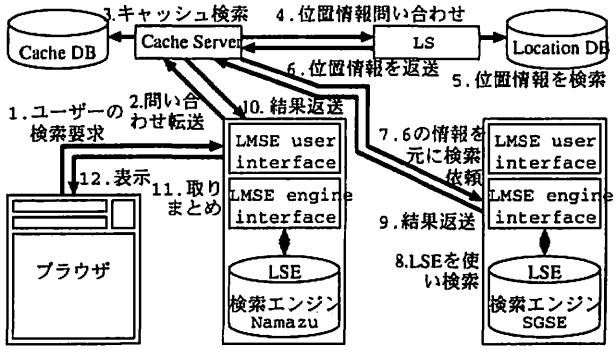

Fig. 1. 現在の CSE の構成と動作

ユーザーの検索要求は入力フォーム付の Web ページに対する入力としてブラウザから LMSE に渡される。以下では、LMSE_0 がユーザーの検索要求である検索論理式 E を受けた後、検索が行われる様子を見て行く。

1. LMSE_0 はユーザーの検索要求を受け取る。
2. LMSE_0 は受け取った検索論理式 E を最適化し、キャッシュサーバ (CS) に E を転送する。
3. CS は E に対する検索結果がキャッシュされているか調べ、キャッシュにあれば 10 を実行し、キャッシュに無ければ 4 を行う。
4. CS は E を LS に送る。
5. LS は位置情報を検索する
6. LS は位置情報の URL を返送する。
7. CS は 6 で得られた URL に示される LMSE に検索を依頼する。

8. 検索を依頼されたLMSEはLSEを使ってEに適合する文書を検索する。

9. 検索を依頼されたLMSEは検索結果スコアの降順にソートしてをCSに返送する。

10. CSは9の結果をキャッシュに格納し、$LMSE_0$にソートした結果を返送する。なお、7から9までは並列に実行され、初回の結果表示に必要な検索結果が揃った時点で$LMSE_0$に検索結果が返送される。残りの検索結果はバックグラウンドでキャッシュに格納される。

11. $LMSE_0$は結果を取りまとめ、ユーザーのブラウザに表示する。

キャッシュサーバが存在しない場合、「次の10件」をユーザーが要求するたびにCGIとして実装されたLMSEが起動され、無視できないオーバーヘッドを生じる。CSEではキャッシュを用いることによって、「次の10件」を表示する際の効率を高め、2回目以降の検索を高速化している。しかし、検索結果を検索式ごとに一度に全てキャッシュするため、キャッシュの利用効率が悪い、キャッシュサーバの負荷が高くなるなどの問題がある。一般にユーザーは少ないキーワードで検索を始め、徐々にキーワードを追加して絞り込むことが多いため、検索結果の件数が多い検索初期にはキャッシュされている検索結果の多くは無駄になってしまう。また、将来CSEを大規模化する際にLSの負荷が問題となると考えられる。

3.2 LSの階層化とCSのLSへの統合

3.1の問題点を解決するために、現在CSE全体に1つしか存在しないLSを、負荷分散のため複数存在させ、CSをLSに統合するとともにキャッシュアルゴリズムを変更してキャッシュ時の負荷を軽減することにした。LSを複数存在させる場合であっても、データベース更新時などにCSE全体を制御する必要があり、各LS毎に位置情報データベースが矛盾するのは好ましくないので、LSはLS_0をルートノードとする木構造を成す。レベルnのLS_nはLS_{n+1}に対してキャッシュサーバとして動作する。LMSEはLS_nの葉ノードとなる。

Fig.2に階層化されたCSEの構成図を示す。この図に示す通り、LMSEはどのレベルのLSの葉ノードとなることが可能である。

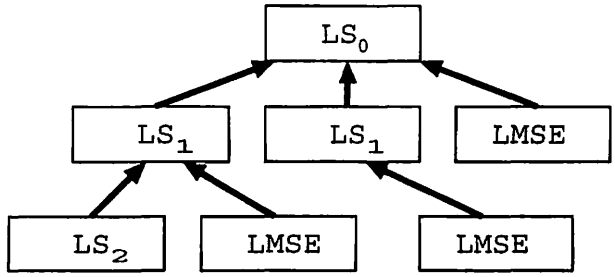

Fig. 2. 階層化されたCSEのLSとLMSEの関係

4 階層化LSの動作

4.1 検索時の動作

LSにおいて、検索と先読みキャッシュの管理をするアルゴリズム**Search** を以下に示す。アルゴリズム**display** とアルゴリズム**lookahead** は**Search** のサブルーチンとして機能する。**display** は検索結果の表示、**lookahead** は検索結果をキャッシュに取り込み、スコアの降順にマージソートする。

アルゴリズム Search

1. $LMSE_0$はユーザーからの検索要求(検索式E)を受け取る。

2. $LMSE_0$はLS_nにSearch(N_{start}, N_{ipp}, E)を送信する。ここで、N_{start}は表示を開始する件数、N_{ipp}は1ページ当たりの件数で、Searchの結果はEに該当するURLの一覧のN_{start}から$N_{start}+N_{ipp}-1$までのN_{ipp}件である。

3. Searchリクエストを受信したLS_nはEに対するキャッシュが存在しない場合は以下のことを行う。

   (a) LN_nはLS_{n-1}に対してAsk(E)を再帰的に送信する。

   (b) LN_nはLS_{n-1}からN_{total}、$\{idf_k\}(k \in K)$、$\{LMSE_i\}(LMSE_i \in S)$、$\{[tf_{max,i,k}, tf_{min,i,k}]\}(k \in K, LMSE_i \in S)$を受け取る。ここで、$N_{total}$は全文書数、$K$は$E$中のキーワードの集合、$idf_k$はキーワード$k$の$idf$値、$S$は$E$に該当する文書をもつLMSEの集合、$LMSE_i$は$LMSE_i$のURL、$tf_{max,i,k}$と$tf_{min,i,k}$は$LMSE_i$の全文書における$k$の出現頻度の最大値と最小値である。

(c) LS_n は $LMSE_i$ のキャッシュとしてキュー Q_i 及び Q_{sorted} を作成する。ここで、Q_i はスコアのリスト $s_i[]$ と URL のリスト $u_i[]$ を含み、先頭ポインタ h_i と末尾ポインタ t_i で制御される。Q_{sorted} も同様である。

4. $|Q_{sorted}|$ を Q_{sorted} の中の要素数とするとき、以下のことを行う。

(a) {display(N_{start}, N_{ipp}, E)} if $|Q_{sorted}| \geq 2N_{ipp}$,

(b) par {display(N_{start}, N_{ipp}, E);lookahead($N_{start} + |Q_{sorted}|$, $2N_{ipp} - |Q_{sorted}|$, E, S, N_{total}) } if $|Q_{sorted}| \geq N_{ipp}$,

(c) {lookahead($N_{start} + |Q_{sorted}|$, $2N_{ipp} - |Q_{sorted}|$, E, S, N_{total}); display(N_{start}, N_{ipp}, E)} otherwise.

5. 終了

アルゴリズム display(N_{start}, N_{ipp}, E)

for $i = 1$ to N_{ipp} do begin

{u, s}=dequeue from Q_{sorted}

ここで、u は URL、s はスコア

U と s を表示

end

アルゴリズム lookahead(N_{start}, N_{ipp}, E, S, N_{total})

1. LS_n は全ての $LMSE_i \in S$ に対して並行に以下のことを行う。

(a) $LMSE_i$ に Search($t_i + 2$, N_{ipp}, E, $\{k, idf_k\}$, N_{total}) を送り、$t_i + 1$ からの N_{ipp} 件を要求する。

(b) $LMSE_i$ は LSE_i に Search($t_i + 2$, N_{ipp}, E, $\{k, idf_k\}$, N_{total}) を送る。

(c) LSE_i は URL 文書の集合 U_i の各要素 u_d のスコア $s_{d,E}$ を $tf_{d,k} \cdot idf_k$ に基づいて計算し、降順に並べたときの $t_i + 1$ から N_{ipp} 件を $LMSE_i$ に返す。ここで、$tf_{d,k}$ は u_d 中の k の出現頻度である。

(d) $LMSE_i$ は結果の N_{ipp} 件を LS_n に送信する。

(e) LS_n は受信した N_{ipp} 件を Q_i に追加する。

2. $s_{min} = \max s_i[h_i]$ なる Q_j から先頭要素を取り去り、Q_{sorted} へ追加する。

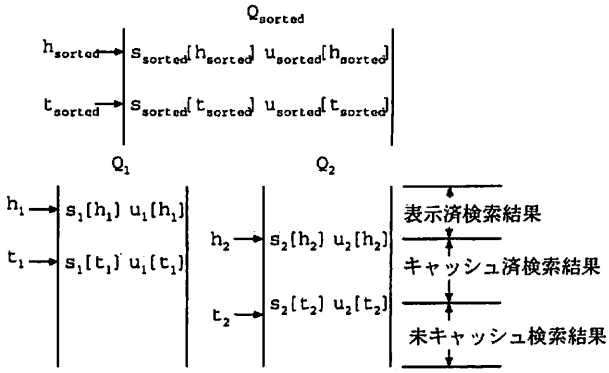

Fig. 3. キャッシュの構造

LS の検索結果を保管するキュー Q_i は LMSE 毎に存在し、Fig.3 のような構造をしている。Q_i に検索結果全体が常に入っている訳ではなく、必要に応じて各 LMSE で検索が行われ、その結果が Q_i に追加される。アルゴリズム**Search** に記述されているように、最低でも表示可能な検索結果が N_{ipp} 件キャッシュされていると「次の 10 件」を表示する際に高速な動作が可能である。キャッシュを消去する条件は特に定めないが、「次の 10 件」を表示する時以外に、同じ検索式が使われることは少ないと思われることから、キャッシュの寿命は長くても数時間程度で十分であると思われる。

4.2 位置情報更新時の動作

位置情報更新時には、LS が主体となって動作する。これは全体を制御するのに都合が良いからである。LS_0 の枝である各 LS に位置情報更新要求を出し、再帰的に位置情報を更新することを想定している。しかし、あるドメイン内の文書のデータベースだけを更新することを目的として、任意の LS が位置情報更新をはじめることも可能である。新しい LMSE を LS に追加登録することを主な目的として、LMSE が主体となって位置情報を更新することもできる。LS は LMSE の自己申告にしたがって LMSE を登録する。現在のところ、認証機能はない。

1. LS は下位の LS に位置情報更新要求を出す。自分自身の枝として LMSE が存在する時には、各 LMSE にインデックスの更新とキーワード、スコア情報の返送を要求する。

2. 位置情報更新要求を受けたLSは下位のLSに位置情報更新要求を中継する。

3. 各LMSEはインデックスを更新し、キーワードとスコアに関する情報を抽出する。

4. 各LMSEはインデックスの更新が終ったらキーワードとスコアの情報をLSに返送する。

5. LSはキーワード、スコアを位置情報として登録し、上位のLSが存在する時はこれらの情報を送信し、LMSEの問い合わせに答えられるようにする。

5 CSEで用いるプロトコル

CSE で用いている通信プロトコル (Cooerative Search Protocol、CSP) は、GMTP(Generic Message Transfer Protocol)[12] の上に構築されている。このため、本稿では特にCSP/GMTPと記述する。GMTPは1本のコネクション型の通信路において双方向に汎用メッセージの送受信を可能とする。引数は任意の文字列で、サイズに制限はなく、一つのメッセージに任意の個数の引数を含めることができる。GMTPは目的に応じてメソッドとその応答 (すなわちAPI) を定義することで特定用途に特化することができる。GMTPにはファイアーウォールを越えたり、CGIとして実装されているLMSEをLSが起動するときのためにHTTPの上にピギーバック形式で送られる、GMTP over HTTPも定義されている。

CSP/GMTPは、実際にはGMTP上に定義されたメソッドとその応答である。以下では、GMTP上のCSE用APIを定義する。本稿での定義は分散tf·idf法によるスコアの計算を実現することを主な目的として、文献[12]での定義を更新するものである。なお、GMTPの定義には変更はない。便宜上、下記のような疑似RPC形式で表す。

$$(y_1, y_2, .., y_m) = Method(x_1, x_2, .., x_n)$$

CSEにキャッシュサーバを導入した[14]でのプロトコルの定義は $tf \cdot idf$ 法を一切考慮していない。本稿では検索時キャッシュのCSEへの導入と、$tf \cdot idf$ 法を利用するためのプロトコルを再定義する。

以下にLSが受け付けるメソッドの一覧を示す。voidは引数、帰り値がないことを表す。

AskMe : `(void) = AskMe(void)`

Callbackを実現するための何もしないメソッド。

Ask : `(Ntotal, Hosts, idfs) = Ask(Expr)`

検索論理式`Expr`に応答できると期待されるLMSEの集合`Hosts`を返す。`Ntotal`はCSE全体の登録済文書の数である。

Search : `(URLs) = Search(Nstart, Nipp, Expr)`

検索論理式`Expr`を検索した結果を、`Nstart`件目から`Nipp`件だけ返す。なお、LMESが受け付けるSearchメソッドとは引数が違う。

Update : `(void) = Update(LMSE, LMSEnumdocs, weightKeys)`

URL `LMSE`で表されるLMSEから、重み付けキーワード集合`weightKeys`を受け取り、サイト集合を完全更新する。`LMSE`に関する情報は一旦削除され、改めて登録される。`LMSEnumdocs`はLMSEに登録されている文書の数である。

UpdateDiffs : `(void) = Update(LMSE, LMSEnumdocs, weightKeys)`

URL `LMSE`で表されるLMSEから、重み付けキーワード集合`weightKeys`を受け取り、サイト集合を更新する。`Update`と違い、差分のみを送る。`LMSEnumdocs`はLMSEに登録されている文書の数である。

UpdateMe : `(void) = UpdateMe(server)`

位置情報更新を開始する要求。上位のLSから送られる。このメソッドを受け付けたLSは下位のLSが存在する時はUpdateMeを下位のLSに送り、LMSEにはUpdateMeを送る。下位のLSとLMSEの位置情報更新が終った後、serverで示されるURLに対してUpdateを送る。

Notify : `(void) = Notify(status)`

`status`で示される状態を通知する。`status`には以下のものがある。

InvalidCache `KeywordList` Update、UpdateDiffs、TransLocationによってLSの位置情報キャッシュが無効になった。この通知を受

けたLSはKeywordListで示される各キーワードの位置情報キャッシュを削除する。

TransLocations : (void) = TransLocations(LMSElist, LocationList)

LSからLSに位置情報を転送する。Update、UpdateDiffsでも転送可能であるが、効率の問題から別のメソッドを定義する。

以下にLMSEが受け付けるメソッドの一覧を示す。

Search : (URLs) = Search(Nstart, Nipp, Expr, idfs)

検索論理式Exprを検索した結果を、Nstart件目からNipp件だけ返す。LMSEで$tf \cdot idf$法によるスコア計算を分散して行うため、LSからidfの値をLMSEに転送する必要があるため、引数にidfsが追加されている。

UpdateMe : (void) = UpdateMe(server)

serverが示すURLに対してUpdate要求を送る。LSEのインデックスの更新を行うことが期待される。

上のメソッドの説明に出て来る引数、帰り値の形式のうち、新しいものを以下に示す。KeywordListは以下のような形をしている。

```
KeywordList = 1*Keyword
Keyword = 1*(ALPHA | DIGIT | EUC)
```

ここで、Keywordは任意の検索語一つを表す。
LMSE集合と位置集合は以下のように表す。

```
LMSElist = 1*(LMSEnum LMSEnumdocs URI)
LocationList = 1*ScoreLine
ScoreLine = Keyword 1*ScoreInfo CRLF
ScoreInfo = LMSEnum ":[" TfMax ":"
                        TfMin "]:" NumDocs
LMSEnum 1*DIGIT
LMSEnumdocs = 1*DIGIT
TfMax = 1*DIGIT
TfMin = 1*DIGIT
NumDocs = 1*DIGIT
```

ここで、URIはRFC2368で定義されたURIである。LMSEnumはLSに登録されているLMSEに付けられた一意の番号で、メソッド毎にその場限りの値として使われる。LMSEnumdocsはLMSEに登録されている文書の総数、TfMax、TfMinはKeywordに対する各文書のスコアの最大値と最小値、NumDocsはURIで示されるLMSEにおいて、Keywordを含む文書の数である。

6 まとめと今後の課題

本稿では協調サーチエンジンにおいて、LSの階層構造化と検索時の先読みキャッシュをLSに追加するためのアルゴリズムについて述べた。しかし、以下のような課題がある。

- 位置情報を更新するときの詳細な動作。
- 負荷分散を考慮した検索時の動作を提案できたが、ルートノードとなるLSの負荷がどの程度軽減できるか検証する必要がある。

参考文献

[1] 馬場 肇『日本語全文検索エンジンソフトウェアのリスト』 http://www.kusastro.kyoto-u.ac.jp/~baba/wais/other-system.html

[2] 山名早人『*Trends of WWW Search Engines*』 http://www.etl.go.jp/~yamana/Research/WWW/survey.html (1998)

[3] 『次世代分散型情報検索システムに関する調査研究報告書』 http://www.jeida.or.jp/committee/jisedai/top.html (1997)

[4] NTTソフトウェア研究所 Ingrid 研究プロジェクト『*Ingrid:* 広域情報検索システム』 http://www.ingrid.org/

[5] 高林 哲 『全文検索システム *Namazu*』 http://openlab.ring.gr.jp/Nnamazu/

[6] Steve Kirsch: Infoseek's approach to distributed search, Report of the Distributed Indexing / Searching Workshop, http://www.w3.org/Search/9605-Indexing-Workshop/Papers/Kirsch@Infoseek.html

[7] Inktomi, http://www.inktomi.com/

[8] FreshEye, http://www.fresheye.com/

[9] 國頭吾郎、相澤清晴『モバイルエージェントを用いた *WWW* 検索システムの協調に関する検討』DICOMO'99 (1999)

[10] 國頭吾郎、相澤清晴、森川博之、宵山友紀『モバイルエージェントを用いた連携検索システムの実装に関する検討』DICOMO'2000 (2000)

[11] 山本崇、佐藤永欣、西田喜裕、上原稔、森秀樹『協調サーチエンジンの研究』,DICOMO'99 (1999)

[12] 西田喜裕、山本崇、佐藤永欣、上原稔、森秀樹『分散サーチエンジンにおける協調型検索』,SWoPP'99 (1999)

[13] 佐藤永欣、山本 崇、西田喜裕、上原稔、森秀樹『協調サーチエンジンにおける *tf·idf* 法に基づく分散スコアリング』DPSWS '99 (1999)

[14] 西田喜裕、山本崇、佐藤永欣、上原稔、森秀樹『キャッシュを用いた協調サーチエンジンの高速化』,DICOMO'2000 (2000)