協調サーチエンジンにおける大域的共有キャッシュ

佐藤 永欣　上原 稔　酒井 義文　森 秀樹

東洋大学工学部情報工学科

E-mail: jju@ds.cs.toyo.ac.jp, {uehara,sakai,mori}@cs.toyo.ac.jp

協調サーチエンジン (Cooperative Search Engine, CSE) はインデックス更新間隔の短縮化を主眼とした分散型サーチエンジンである。CSE では文書収集、インデックス更新をボトムアップで行うため、更新所要時間を大幅に短縮できるが、検索時の効率が悪いのが難点である。WWW 検索では 10 件毎に検索結果を表示し、検索結果の続きを「次の 10 件」等のボタンを押す事で得る場合が多い。我々はこれを継続検索と呼んでいる。継続検索時に検索対象とするサイトを絞り込むことで、継続検索の 2 ページ目以降での検索効率の悪さは改善した。しかし、CSE ではどのサイトでヒットするかは検索するまで解らないため、1 ページ目の検索では効率が悪いままであった。本稿では、CSE においてキャッシュを大域的に共有することで 1 ページ目の検索を効率化する手法について提案する。

Global Shared Cache in Cooperative Search Engine

Nobuyoshi Sato, Minoru Uehara, Yoshifumi Sakai, Hideki Mori

Department of Information and Computer Sciences, Toyo University

Cooperative Search Engine (CSE) is a distributed search engine which is developed in order to reduce updating interval times. In CSE, collecting documents and updating indexes are done in bottom up. Therefore, CSE realizes very short updating interval times. However, CSE is inefficient to search documents. In WWW information retrieval, results are often divided in 10 items in a page, and rests of them are displayed by pushing "next 10" bottom. We call this continuous searching. In after second pages of continuous searching, the performance of CSE is improved by score based site selection. However, we cannot apply score based site selection first page retrieval. So, CSE's first page retrieval is still inefficient. In this paper, we describe how to retrieve first page efficiently by using global shared cache.

1　はじめに

インターネットの発展に伴い膨大な Web ページの検索が重要な課題となっている。中でもサーチエンジンのインデックス更新期間を短縮することが重要となっている。そこで、我々は分散型アーキテクチャによる協調サーチエンジン (Cooperative Search Engine, CSE) を開発した。CSE では、更新を各 Web サーバで局所的に行い、それらを複数のメタサーチエンジンで統合することで 1 つの大域的サーチエンジンを構成する。CSE は更新時間の大幅な短縮を実現した [6]。

CSE では、更新時間の短縮の代償として検索時間が増大する。イントラネットではこの増分も無視できるが、インターネットでは無視できないほど大きくなる。国際的な企業のイントラネットはインターネットに匹敵する。そこで、我々はいくつかの技法により検索時間の縮小を試みてきた。まず、クエリに基づくサイト選択を行った [11]。この技法では「誰がそれを知っているか」という知識 (Forward Knowledge) に基づいて、全く知識を持たないサイトへは問い合わせない。具体的には、ブーリアン検索において語 A、B を知るサイトの集合をそれぞれ S_A、S_B とすると、A and B は $S_A \cap S_B$、A or B は $S_A \cup S_B$、A not B は S_A に限られる。これはイントラネットでは極めて有効に働く。本学では Web サーバは 22 サイト存在するが、平均して 1.6 台にしか問い合わせないため、不要な問い合わせを約 93%削減した。しかし、インターネットのように 100 万を超えるドメインが存在する場合、9 割削減しても 10 万サーバへ問い合わせなければならない。

次に、継続検索 (いわゆる「次の 10 件」の検索) に注目した。本学プロキシサーバのログを解析したところ 7 割が継続検索を行っていた。文献 [3] によれば継続検索で必要なのは最初の 2、3 ページである。これに

より不要なページの送信を省くことができる。CSEでは、継続検索実現のためにキャッシュサーバ(CS)を導入したが、この際、先読みを行う。先読みキャッシュ方式[7]では、常に1つ先のページを先読みしておくことで、検索時にキャッシュされた内容を直ちに返し、みかけの応答時間を短縮する。この結果、イントラネット規模では他のサーチエンジンと同等の性能が得られるようになった。

また、継続検索時にスコアに基づくサイト選択[8]を行った。これはスコアに基づく検索の遅延評価である。各サイトはクエリに対する未返答の最高スコアを返し、そのスコアの文書が必要になった時点でその文書を含むページを返す。これにより、インターネットのように10万サーバへ問い合わせなければならない場合でも継続検索の2ページ目以降はたかだか10サイトに問い合わせるだけで十分である。しかし、継続検索の1ページ目を求めるには10万サイトへ問い合わせる必要がある。ただし、実際にはブーリアンクエリのスコアを予測して逐次検索するためそれほど多くはならない。さらに、単独キーワードの場合は正確に20サイトを限定できる。しかし、1ページ目の検索はCSごとに発生するため、同じクエリが100万組織で発生すると10万×100万の検索が輻輳してしまう。そこで、本文では、継続検索の1ページ目におけるサイト選択をキャッシュの共有により最小とする方式を提案する。これにより同じクエリの送信を10万×100万から10万×1へ削減できる。

本文の構成は以下のとおりである。第2章では関連研究について述べる。第3章では現在のCSEについて述べる。第4章ではCSEに大域的共有キャッシュを導入する。第5章では評価について述べる。最後に結論を述べる。

2 関連研究

情報検索の結果のキャッシュではないが、WWWではトラフィック削減、応答時間の改善などを目的として、以前からキャッシュの利用が行われてきた。以前はApache等の一部のWWWサーバのキャッシュ機能を用いることが多かったが、最近は階層的にプロキシサーバを配置し、Internet Cache Protocol (ICP)[12]によるキャッシュ共有が主流である。現在、ICPによるキャッシュ共有を行えるWebプロキシサーバには、Squid[10]、Inktomi Traffic Server[5]などがある。

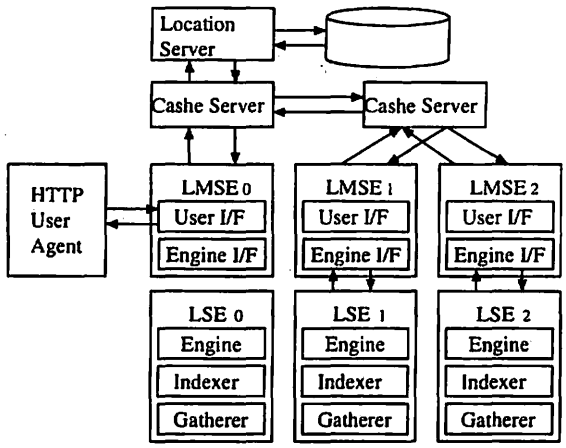

Fig. 1. CSEの構成と概要

ICPはWWWプロキシサーバ間でキャッシュの情報をやりとりするために用いられるプロトコルである。ICPはキャッシュオブジェクトの有無を問い合わせる等に使われ、実際のキャッシュオブジェクトの転送にはHTTPが使われる。ICP自体は単純なプロトコルで、キャッシュが存在するかどうかの問い合わせる機能しかない。ICPによるキャッシュ共有では近くにあるプロキシサーバ同士の情報交換が基本となる。

コロラド大学のHarvest Projectでは Harvest Broker[2]やHarvest Gathererのような検索システムと一体で、Harvest Cacheを開発した。Harvest Cacheは現在一般的に使われているSquidの元となったWWWキャッシュソフトウェアで、ICPをサポートしている。Harvest CacheはHarvest BrokerやGatherer等と一体のシステムをなすが、情報検索の結果をキャッシュしておくソフトウェアではなく、WWWのトラフィック削減を主眼に開発された。

3 協調サーチエンジン

CSEは図1に示されるような部品から構成される。

- Location Server (LS): LSはFK(Forward Knowledge)を一元管理するサーバである。LSはCSE全体でただ一つだけ存在する。
- Cache Server (CS): CSはFKと検索結果をキャッシュするサーバである。LSはトップレベルのCSとも考えられる。検索結果をキャッシュすることで継続検索(次の10件の検索)を実現する。また、CSは後述のLMSEを並列に呼び出し、並列検索を行う。
- Local Meta Search Engine (LMSE): LMSEは、ユーザからの要求を受け付けCSに転送したり(図

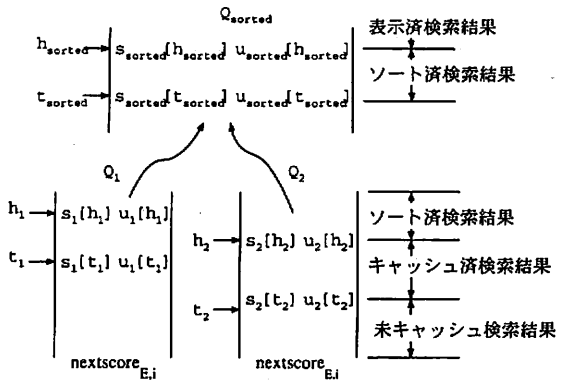

Fig. 2. CS のキャッシュの構造

1 の User I/F)、後述の LSE を呼び出し局所的な検索をしたり (図 1 の Engine I/F) する。LSE の差異を吸収するメタサーチエンジンである。

- Local Search Engine (LSE):LSE は局所的な文書収集 (図 1 の Gatherer)、インデックス作成 (図 CSE の構成と概要の Indexer)、検索 (図 1 の Engine) を行う。CSE では Namazu[1] や SGSE[9] を LSE として使用できる。

図 2 に CS におけるキャッシュの構造を示す。図 2 で、Q_{sorted} はソート済みキャッシュ、Q_1、Q_2 は $LMSE_1$ と $LMSE_2$ に対応する未ソートキャッシュである。$LMSE_n$ から受け取った検索結果は Q_n に一端格納される。Q_n の内容は LMSE によってソートされており、マージソートされて Q_{sorted} に格納される。Q_{sorted} に格納できなかったあまりは Q_n に留まる。

4 大域的共有キャッシュ

CSE ではユーザからの検索要求を各 LMSE が受け付け、最寄りの CS に問い合わせる。このためキャッシュが各所にでき、同じクエリの送信が繰り返されていた。そこで、同じクエリが発生したときに最初に検索を始めた CS に依頼することでキャッシュを共有する。

しかし、単にキャッシュを共有するだけでは、負荷分散が問題となる。インターネット規模の検索要求が、それを前提としていない CS に集中すると DoS 攻撃を受けたような状態になってしまう。したがって、キャッシュ共有と負荷分散を両立するには、キャッシュをコピーする必要がある。CS は検索結果のキャッシュを持ち、LS はサイト選択のキャッシュを持つ。サイト選択のキャッシュには、LMSE 集合と最初にクエリを発した CS を登録する。キャッシュ共有のアルゴリズムは以下の通りである。

1. ユーザからのクエリ E を受けた $LMSE_0$ は CS_0 に標準化された E' を送る。
2. CS_0 はキャッシュを調べ、あればそれを返すか、継続検索を行いその結果を返す。なければ LS に E' を送り、サイト選択を依頼する。
3. LS はキャッシュを調べ、E' が未検索なら、CS_0 を E' と関連付け、E' に基づきサイト選択を行い、LMSE 集合を返す。また、E' が CS' で検索済みなら、CS_0 にサイト選択の結果として CS' を返す。
4. CS_0 はサイト選択の結果 S として LMSE 集合を受け取れば、S の各 LMSE に並行に E' を送信する (5 へ)。また、$S = \{CS'\}$ であれば CS' に E' を送信する (7 へ)。
5. LMSE は LSE を用いて検索を行い、結果を CS_0 に返す。
6. CS_0 は各 LMSE からの検索結果をまとめランキング順に並び替える (9 へ)。
7. CS' は E' に対する検索結果を CS_0 に返す。
8. CS_0 は CS' からの結果を自らのキャッシュにコピーする。
9. CS_0 は $LMSE_0$ に検索結果を返す。サイト選択の結果 CS が返されるのは、最初の 1 ページ目を検索するときだけである。2 ページ目以降は規模の大小にかかわらず 10 サイトにしか問い合わせないため、LMSE に問い合わせることで負荷分散を実現する。

4.1 CSE で用いられるプロトコル

CSE で用いる通信プロトコル CSP (Cooperative Search Protocol) は TCP 上に GMTP (Generic Message Transfer Protocol) という独自の RPC 指向メッセージ通信プロトコルの上に構築されている GMTP は LS、CS、LMSE の各コンポーネント間の通信に使用される [4]。また、GMTP のメッセージを HTTP 上のデータとして送る GMTP over HTTP も定義されており、ファイヤーウォールに特別な設定をすることなしにファイヤーウォールを越えることが可能である。LMSE は CGI として実装されているため、LMSE にリクエストを送るには GMTP over HTTP が必須である。ただし、GMTP over HTTP を用いる際には受信側が CGI であるため一回の GMTP セッションでは要求とその応答の一回のメッセージ交換しかできない。

GMTPには、要求メッセージ(Request)と応答メッセージ(Response)の2種類のメッセージがある。はじめに要求メッセージの形式について述べる。

Request = CommandLine * ParameterLine

要求メッセージは、1行以上のテキストで、最初の行が要求の種類を表すCommandLineで、残りの行が引数を表すParameterLineである。なお、1行中の文字数は1000文字以内とする。

```
CommandLine = "REQ" 1*SP Method 1*SP
ArgNum 1*SP CRLF
Method = ALPHA *62(ALPHA | DIGIT)
ArgNum = 1*DIGIT | "+"
ParameterLine = "+" LINE CRLF
    | "-" LINE CRLF
    | ";" LINE CRLF
    | "," LINE CRLF
    | "." LINE CRLF
    | SP LINE CRLF
LINE = <any OCTET except CR and LF>
```

ここで、Methodはメソッド名を表す63文字以下の名前であり、その種類は後述する。

ArgNumは引数の個数を表す数値または文字'+'である。'+'は1個以上の可変長引数を意味するが、定数個の引数を要求するメソッドに対して指定することはできない。'+'の定義から可変長引数のメソッドであっても、引数が0個の場合は'+'は使用できない。

LINEは改行文字を含まない任意の文字列である。ParameterLineの先頭1文字には以下の意味がある。

"+" この引数は改行を伴い次行に続く

"-" この引数は改行を伴わずに次行へ続く

";" この引数は改行を伴い本行で終わる（次行は次の引数となる）

"," この引数は改行を伴わずに本行で終わる（次行は次の引数となる）

"." この引数は改行を伴い本行で終わり、次行はない

SP この引数は改行を伴わず本行で終わり、次行はない

次に応答メッセージの形式について述べる。

```
Response = StatusLine *ParameterLine
StatusLine = Status 1*SP Method 1*SP
                        ArgNum 1*SP CRLF
Status = 3DIGIT
```

Table 1. Status及びMethodの意味

Status	Method	意味
100	メソッド名	割り込み
200	OK	成功(応答後継続)
210	Quit	成功(応答後切断)
300	ServerNotFound	接続失敗
310	NotConnect	接続失敗
320	AccessDenied	接続失敗
400	IllegalRequest	失敗
410	UnknownMethod	失敗
420	IllegalParameter	失敗
430	TimeOut	失敗
440	Illegal Expression	失敗
500	Internal Error	失敗

応答メッセージも1行以上のテキストで、1行目は要求の結果を表すStatusLine、残りの行が引数を表すParameterLineである。1行中の文字数は1000文字以内である。StatusLineには、要求の結果を3桁の数値で示したStatusと、それに対応した名称Methodと、結果の値を示すParameterLineの数であるArgNumがある。Status、Methodの値と意味を表1に示す。

表1で、割り込みは受信側が要求に答える前に送信側へ問い合わせをすることである。要求メッセージと応答メッセージは先頭行の先頭要素（"REQ"とStatus）が異なるだけで対称的な形式を持つ。そのため受信者が送信者に対して問い合わせをするときには割り込みを用いて要求メッセージを送ることができる。

GMTPは目的に応じてメソッドとその応答からなるAPIを定義することで特定の用途に特化できる。CSEでは協調検索のためのAPIを定義しているが、拡張を繰り返したため、CSEのコンポーネント毎にメソッドの引数が少しずつ違う。このため、大域共有キャッシュの導入にあわせて整理する。ここでは、便宜上以下のような疑似RPC形式で表す。

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

以下にCSEで用いられるGMTP上のAPIを示す。CSEのコンポーネントによって受付けるメソッドが違うが、処理できないメソッドを受信した場合はGMTPのエラーメッセージ410(UInknown Method)を返す。

Update : (void) = Update(LMSE, LMSENum-Docs, weightKeys)

LSが受け付ける。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。LMSENumDocs は LMSE に登録されている文書の数である。

Ask : (LmseHosts, wholeNumDocs, idfs, CacheHosts) = Ask(Expr)
LS が受け付ける。検索論理式 Expr に応答できると期待されるサイトの集合 LmseHosts を返す。wholeNumDocs は CSE 全体の登録済文書の数である。Expr に関するキャッシュを持つ CS がある場合にはその CS を CacheHosts として返す。

Search : (URLs) = Search(Expr, ShowPos)
CS と LMSE が受け付ける。検索論理式 Expr の検索結果がキャッシュにあるかどうかを調べ、あればキャッシュから検索結果を返送する。無ければ Expr を持つ LMSE に検索を依頼し、結果をキャッシュするとともに検索結果を返送する。返送される検索結果はスコアの降順にソートされ、ShowPos で指定される位置の検索結果のみが返送される。なお、検索結果の件数が、ShowPos で指定した件数に満たない場合は、検索結果として得られた件数のみが返送される。

TransCache : (CacheItems) = TransCache(Expr)
CS が受け付ける。Expr で示される検索式を検索した結果のキャッシュの内容 CacheItems を送信するように要求する。CS は原則としてキャッシュしている Expr の検索結果を全て返答する。これにはソート済みキャッシュと LMSE 毎の未ソートキャッシュが含まれる。キャッシュが存在しない場合は CacheItems の代わりに文字列 "Not Found" が返される。

UpdateMe : (void) = UpdateMe(Server)
LS が受け付ける。Server が示す URL に対して Update 要求を送る。

AskMe : (void) = AskMe(void)
コールバックを実現するための何もしないメソッド。このメソッドは GMTP over HTTP を使う必要がある CGI プログラムの起動を要求する目的で用意された。

以下に API の定義で用いられた引数、帰り値の説明を述べる。サイト集合 LmseHosts は以下のように表す。ここで、URI は RFC1945 で定義された URI である。

```
LmseHosts = *(URI weightKeys CRLF)
```

LMSENumDocs は以下のような形式をしている。

```
LMSENumDocs = 1*DIGIT
```

CacheHosts は以下のような形式をしている。

```
CacheHosts = *(URI Expr CRLF)
```

ここで、Expr は通常、ユーザーから与えられた検索式で、URI で示される CS には Expr に関するキャッシュが格納されている。将来的に、与えられた検索式の部分式のキャッシュも共有するすることを考慮した仕様である。

ShowPos はソート済み検索結果の何件目から何件を返答するかを指定する。

```
ShowPos = (BeginItem SP NumItem)
   BeginItem = 1*DIGIT
   NumItem = 1*DIGIT
```

BeginItem は返答を開始する検索結果が先頭から何番目であるかを示し、NumItem は何件返答するかを示す。NumItem が 0 の時は BeginItem 番目から残り全ての検索結果を表示することを意味する。BeginItem が 0 の場合は暗黙のうちに 1 番目から表示すると解釈される。

CacheItems はキャッシュの内容である。

```
CacheItems = *((URI | "Sorted" ) URLs
                                  CRLF)
```

キャッシュの内容は文書を所有する LMSE の URI またはソート済みキャッシュを示す文字定数 "Sorted" と、検索結果を含む URLs の組である。

URL 集合 URLs は以下のように表す。

```
URLs = "totalmatch:" 1*DIGIT CRLF CRLF
   *("title:" TEXT CRLF
     "score:" 1*DIGIT CRLF
     "url:" URI CRLF
     "summary:" TEXT CRLF
     CRLF)
```

tolalmatch の行は全体のヒット数を表している。また、title の行はヒットした文書のタイトルを表し、score の行はスコア、url の行は URL を表している。そして、summary の行は概要である。

Table 2. 応答時間の比較

	キャッシュ非共有		キャッシュ
	ヒットあり	ヒットなし	共有
応答時間	0.45 [sec]	4.67 [sec]	0.66 [sec]

5 評価

キャッシュを共有した場合としない場合との応答時間の比較を行った。キャッシュを共有した場合、複数のLMSEに問い合わせる代わりに一ヶ所のCSに問い合わせてキャッシュの内容を転送するだけで済むので、応答時間は改善する。

図2にこの結果を示す。ここでは比較対象として、LMSEから問い合わせを受けたCS_0に検索結果がキャッシュされていた場合の応答時間も測定した。LMSE2台の計算機に5個づつ用意した。CSは2台の計算機に1個ずつ用意した。$LMSE_0$からの検索要求は最初の10件である。なお、用意したLMSEのインデックスは全く同じ内容のものであり、CSの動作としてはワーストケースである。使用した計算機は以下の通りである。

- PentiumIII 700MHz 192MB FreeBSD 4.3-STABLE (CS、LMSE)
- Celeron 333MHz 128MB FreeBSD 4.3-STABLE (LS, CS, LMSE)

図2のヒットあり、ヒットなし、キャッシュ共有はそれぞれ、CS_0に検索結果がキャッシュされていた場合の応答時間、CS_0が複数のLMSEに対して検索要求を送った場合の応答時間、キャッシュを共有した場合の応答時間である。測定はそれぞれ5回行い、平均値を採用した。

キャッシュを共有した場合は1秒以内にキャッシュの転送と表示が完了し、キャッシュを共有しなかった場合に比べてはるかに高速である。しかし、当然のことながら、CS_0に検索結果がキャッシュされていた場合よりは応答に時間がかかる。また、CS_0に検索結果がキャッシュされていれば、キャッシュの共有、非共有にかかわらず応答時間は同じと考えられる。

6 まとめ

検索結果のキャッシュを大域的に共有することで、同じクエリに対する不要な検索を抑制することができるようになった。また、キャッシュの共有は検索時の負荷が高まる1ページ目のみを対象とし、2ページ目以降はCSの負荷分散のため非共有とし、ユーザーの最寄りのCSが各LMSEに対して検索を要求する。

しかし、検索結果のキャッシュが有効なのは次のインデックス更新までで、インデックスが更新されるとキャッシュは無効となる。そのため更新頻度の高いCSEではキャッシュを長時間有効に保てない。今後はキャッシュのうち更新された文書に関する部分だけを無効にする等の方式により、更新後にもキャッシュの有効性を維持する方式を開発する。

参考文献

[1] 馬場 肇、"日本語全文検索システムの構築と活用"、ソフトバンク、ISBN4-7973-0691-2 (1998)

[2] C. Mic. Brown et. al. "The Harvest Information Discovery and Access System", http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/schwartz.harvest/schwartz.harvest.html (1994)

[3] Evangelos P. Markatos: "On Caching Search Engine Query Results", In Proc. of the 5th International Web Caching and Content Delivery Workshop, (2000)

[4] 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, pp.87-92 (1999)

[5] 太田 佳伸, インクトゥミ・ジャパン株式会社監修, "キャッシュ技術入門", セレンディップ、ISBN4-7978-2008-X (2000)

[6] Nobuyoshi Sato, Takashi Yamamoto, Yoshihiro Nishida, Minoru Uehara, Hideki Mori, "Information Retrieval Method for Frequently Updated Information System", Subhash Bhalla(Ed.), "DNIS 2000", LNCS Vol.1966, Springer-Verlag, pp.188-199ISBN3-540-41395-2 (2000)

[7] 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継続検索のための先読みキャッシュ方式", DPS ワークショップ論文集, pp.205-210 (2000)

[8] 佐藤 永欣, 上原 稔, 酒井 義文, 森 秀樹, "協調サーチエンジンにおけるスコアに基づくサイト選択", DICOMO2001, pp.465-470 (2001)

[9] "SonyDrive Search Engine", http://www.sony.co.jp/sd/Search/

[10] "Squid Web Proxy Cache" http://www.squid-cache.org/

[11] 上原 稔, 佐藤 永欣, 山本 崇, 西田 喜裕, 森 秀樹, "協調検索エンジンにおけるクエリ対象の最小化", DPS ワークショップ論文集, pp.85-90 (1999)

[12] D. Wessels, K. Claffy, "Internet Cache Protocol (ICP), version 2", RFC2186 (1997)