協調サーチエンジンにおける組織内文書のセキュリティ

佐藤 永欣　宇田川 稔† 上原 稔† 酒井 義文‡
E-mail: {jju,ti980039}@ds.cs.toyo.ac.jp, uehara@cs.toyo.ac.jp, sakai@biochem.tohoku.ac.jp
† 東洋大学工学部情報工学科
‡ 東北大学大学院農学研究科

集中型アーキテクチャに基づくサーチエンジンでは新鮮な情報検索が困難である。そこで我々は分散型アーキテクチャに基づくイントラネット向けサーチエンジンである協調サーチエンジン (Cooperative Search Engine, CSE) を開発した。イントラネットにおいては組織の構成要素ごとに外部に出すべき情報と内部に閉じ込めるべき情報があるが、CSE ではこれを考慮していなかった。そこで、CSE において文書ごとに公開・非公開を設定し、非公開文書の存在の痕跡を外部に出さずに、組織の構成要素内部からは公開・非公開文書を統合的に検索する手法を提案する。

Security of Restricted Documents in Cooperative Search Engine

Nobuyoshi Sato†, Minoru Udagawa†, Minoru Uehara†, Yoshifumi Sakai‡
†Department of Information and Computer Sciences, Toyo University
‡Graduate School of Agricultural Science, Tohoku University

It is difficult for centralized search engines to retrieve fresh information. So, we have developed a distributed search engine for intranets, Cooperative Search Engine(CSE). In intranets, each section of a organization has two sorts of docuemnts. One is open to the public, the other is private or secret doucments. However, in CSE, this simple secrurity was not considered. In this paper, we propose a method to establish open and private of documents and search them unifiedly form inside of a section without leaving traces to outside.

1 はじめに

近年、組織内情報検索が重要となってきた。組織内情報検索では新鮮な情報検索が必要である。特にビジネス分野では必要な情報が得られないと機会を失うことになり、損失に繋がる。情報検索には一般的にサーチエンジンが用いられる。組織の規模が大きくなると集中型サーチエンジンでは新鮮な情報検索が困難となる。これは、集中型サーチエンジンでは、ロボットで文書を収集し、インデックスを更新するまでに長い時間がかかるためである。したがって、新鮮な情報検索には分散して文書収集、インデックスの作成、更新が可能な分散サーチエンジンが適している。

そこで、我々は、分散型アーキテクチャに基づく協調サーチエンジン (Cooperative Search Engine, CSE) を開発した [1]。CSE は、各 Web サーバに配置された局所サーチエンジンをメタサーチサーバで統合した大域的サーチエンジンである。CSE はボトムアップで文書収集、インデックス作成等の更新作業を行うので、CSE は更新に関してスケーラブルであり、規模にかかわらず短時間でインデックスを更新できる。また、検索が遅いという問題点があったが、検索結果をキャッシュするなどの高速化技法によりある程度の検索時のスケーラビリティを実現できた。

一般に組織は複数の構成要素に分割される。各構成要素間の関係は、木構造などの場合もあるが、構造を持たない不定型の場合もある。CSE は組織の部門ごとにある Web サーバに局所サーチエンジンを配し、組織で一つのメタサーチサーバでそれらを統合することを前提としてきた。また、我々は以前、メタサーチサーバが単一故障点となるため、耐故障性の向上を目的として複数のメタサーチサーバを冗長化する構成を提案し、評価を行った。冗長化されたメタサーチサーバは P2P により同じ情報を共有し、故障に備える。

部門毎に文書・文書に含まれる語の秘匿、公開を制御する機能を CSE に付加する場合に考えられる方法として、メタサーチサーバを部門毎に分離する、またはメタサーチサーバは分離せずに語を暗号化して情報を秘匿するなどがある。以前我々は、メタサーチサーバを階層化する提案を行った [3] が、部門ごとに階層化されたメタサーチサーバを多重化するためには、各階層で 2 台以上のメタサーチサーバが必要になる。また、文書・キーワードの秘匿の制御も、メタサーチサーバ

単位になり、運用、管理が繁雑になる。さらに、メタサーチサーバの階層が深くなるほど、CSE の原理上検索時の遅延が増大する。そこで、本論文ではメタサーチサーバは階層化せずに、語を暗号化によって秘匿する方法で CSE で文書の検索・公開の制御を試みる。

2 関連研究

Web ロボットを用いて文書を収集する集中型サーチエンジンでは、ロボットがアクセス制限の対象であるため、アクセス制限文書の検索は原則としてアクセスが許可されたドメイン内にロボットとサーチエンジンが存在する場合に限られる。イントラネット内、組織の構成単位内のみの文書を検索するには十分であるが、アクセス制限文書と公開文書を統合的に検索するのは難しい。分散サーチエンジンでは、ロボットを使わずに収集すればサーチエンジンがアクセス制御を行えばよいので、統合的な検索が可能である。

組織内情報を分散検索で統合する試みは MultiText[8] で行われている。MultiText はマルチユーザ・マルチサーバ環境のための分散型全文検索システムである。MultiText はクライアントの要求を受理する Marshaller/Dispatcher、文書を保管する Text Server、検索を行う Index Engine で構成され、それぞれ異なるホストで運用できる。

WWW の以外の分散サーチエンジンでは人を探す Whois++[9], 等が代表的である。Whois++で採用された Forward Knowledge は以後の分散検索の基本となった。WWW の分散検索としては Harvest[10]、Ingrid[11] などが挙げられる。

3 協調サーチエンジン

CSE は Fig. 1 に示されるような以下の部品から構成される。

- Location Server (LS): LS は FK (Forward Knowledge) を一元管理する。LS は FK を用いてクエリに基づくサイト選択を行う。LS はサイト選択キャッシュ(Site selection Cache, SC) を持つ。
- Cache Server (CS): CS は、サイト選択の結果と検索結果をキャッシュするサーバである。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。CS は検索結果キャッシュ(Retrieval Cache, RC) とサイト選択キャッシュを持つ。CS のサイト選択キャッシュは

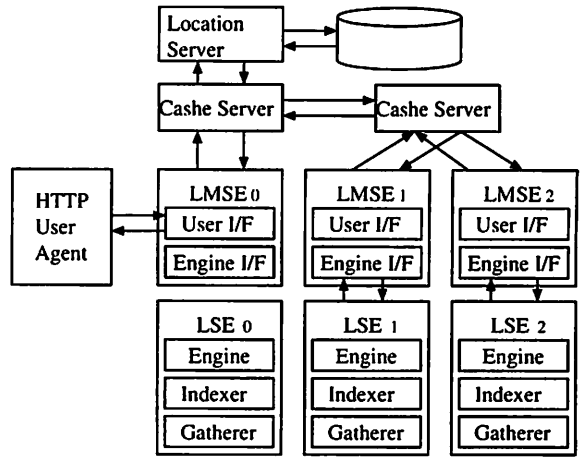

Fig. 1. CSE の概要

LS のサイト選択キャッシュの部分的、不完全なコピーである。CS のサイト選択キャッシュに検索によって生じた変化は LS のサイト選択キャッシュに反映される。

- 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（Fig. 1 の User I/F）、後述の LSE を呼び出し局所的な検索をしたり（Fig. 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（Fig. 1 の Gatherer）、インデックス作成（Fig. 1 の Indexer)、検索（Fig. 1 の Engine）を行う。

更新時の各構成要素の動作は以下の通りである。

1. LMSE は LSE を用いて文書を収集する。
2. LMSE は LSE を用いてインデックスを更新する。
3. LMSE は LS に Forward Knowledge（FK、語の集合、全文書数、語を含む文書数とその最高スコア）を送信する。LS は FK をメタインデックスに記録する。

CSE では、NFS 等のファイルシステム経由で直接収集できる文書はファイルシステムを通して直接収集し、クラスタを用いて並列にインデックス作成を行う事が可能である。また、直接収集できない文書は Web サーバに用意した文書収集用 CGI を用いて一括転送する事も可能である。これらが使用できない場合のみ、通常のロボットによる収集が用いられる。この結果、東洋大学の事例では約 1 分で全文書のインデックスを更新できた。

一方、検索時における各構成要素の振る舞いは以下の通りである。

1. ユーザはブラウザにより身近な $LMSE_0$ に検索を依頼する。
2. $LMSE_0$ は CS に検索を依頼する。
3. もし CS の RC に検索結果の次ページまでキャッシュされていたら 8 へ。もし、該当ページまでしかキャッシュされていなければ 5 へ。
4. CS は SC にサイト選択結果がキャッシュされていたら 5 へ。そうでなければ、LS にクエリに基づくサイト検索を依頼し、スコアの降順に並んだサイトのリストと各語の *idf* を受信する。
5. CS は次ページまでの項目を埋めるのに必要なだけリスト上位からサイト $LMSE_i$ を選び、検索要求を並行に送信する。
6. $LMSE_i$ は LSE に検索を依頼し、検索結果と次の最高スコアを CS に返す。
7. CS は結果をマージし、サイトのリストをスコア順に並び替える。
8. CS は結果を $LMSE_0$ へ返す。
9. $LMSE_0$ は結果を HTML に整形し、ユーザへ返す。

CSE では、検索時に通信による様々な遅延のため応答時間が長くなる。そこで、一回あたりの通信量を極力減らす、不要な通信は行わない等、以下の高速化技法などが提案されている。

クエリに基づくサイト選択 (QbSS) [4]。CSE は積 (AND)、和 (OR)、差 (NOT) の論理検索をサポートしている。ここで、クエリ A、B に対する選択サイトを S_A、S_B とすると、"A and B"、"A or B"、"A not B" はそれぞれ $S_A \cap S_B$、$S_A \cup S_B$、S_A となる。

先読みキャッシュ [3]。「次の 10 件」をバックグラウンドで先読みして応答時間を短縮する。これにより 2 ページ以降の連続した「次の 10 件」検索は常にキャッシュにヒットする。

スコアに基づくサイト選択 (SbSS) [5]。各サイトのクエリに対する最高スコアを収集し、「次の 10 件」検索時にクエリを送信するサイトを多くとも 10 サイトに限定する。これにより論理検索の 1 ページを除く連続した「次の 10 件」検索では、規模に依存せず一定の応答時間を実現した。

永続的キャッシュ [7] 更新間隔の短い CSE ではキャッシュを長く用いることができない。そこで、更新後に再検索を行うことでキャッシュの寿命を永続的にする。これにより一度検索された（論理型）クエリの応答時間も規模に依存せず一定となった。

以上の技法は以下のような場合に適用できる。第一に、検索式が単一キーワードであるか OR 演算子のみを含む場合、または 2 ページ以降の「次の 10 件」検索では、スコアに基づくサイト選択により一定の応答時間を実現できる。第二に、更新前にクエリが検索されていれば、永続的キャッシュにより、一定の応答時間を実現できる。第三に、更新後にクエリが検索されていれば、大域的共有キャッシュにより即座に応答可能である。最後に、それ以外の場合はクエリに基づくサイト選択を行う。クエリに基づくサイト選択による応答時間は検索対象サイトの数に依存し、検索対象サイトの数は一般に CSE の規模に依存する。

4 文書セキュリティの実現

CSE における文書セキュリティは、各ディレクトリの全文書に含まれる各語について、サーバまたはドメインごとに検索可、不可を設定することにより実現する。各組織の構造は木構造になっていることが多いため、サーバやドメインも木構造を構成すると仮定する。組織の構造が不定型の場合でも木構造に当てはめることは可能である。CSE では、LMSE を Web サーバ毎に、CS を概ねドメイン毎に設置することを想定しているが、CS の配置、動作などはドメイン間の親子関係等に依存せず、論理的にはフラットな構造である。文書セキュリティを実現するため、本論文では暗号化に用いられる鍵の流通範囲のみを階層化し、CS などはフラットな構造を許す。CS の階層化は実際の組織の構造にあった配置を可能にするが、検索時には通信遅延が大きくなる。しかし、階層化を暗号鍵に限定することで検索応答時間を維持できる。また、更新時には CS を中継に用いることで LS の負荷を軽減可能であるが、CSE の規模、各 LMSE の更新のタイミング等、場合によっては更新所要時間が長くなる。したがって、今までと同様に、更新時に CS を中継するかどうかは LMSE（または下位の CS）が選択可能とする。

CSE において、検索を行うためには Forward Knowledge として各語を LS に送らなければならない。しかし、アクセス制限されている文書から取り出した語を外部にそのまま流出させるのは問題がある。また、たとえ暗号化したとしても、本来秘匿すべき語を Forward Knowledge の形で外部に出すため、暗号鍵の漏洩が起

きても容易に解読できない方式が望ましい。そこで、MD5 等の一方向ハッシュ関数で語と暗号鍵をハッシュすることにより暗号化し、それらを Forward Knowledge として LS に送ることとする。検索時に暗号化された語を解読する必要が無いように、LMSE、LS、CS の動作を決めなければならない。

Forward Knowledge として LS に送るべき語には長さの制限が無いため、一方向ハッシュ関数を用いる場合、ハッシュが衝突すると本来別の語であったものが、Forward Knoledge のレベルで混同され、検索対象とされてしまう。したがって、CSE の規模に応じてハッシュ値のビット数やアルゴリズムを決める必要がある。また、復号化の必要が一切無いので、暗号化方式・暗号鍵は LMSE が任意に選択できる。

また、一方向ハッシュ関数の代わりに公開鍵暗号を使うことも考えられる。この場合は、鍵のペアの生成後、秘密鍵のみ保持して公開鍵を破棄することにより、復号化を不可能にする。ただし、一般に公開鍵暗号の計算には時間がかかるため、高速な更新が特長の CSE への適用は簡単ではない。

暗号鍵は LMSE 毎、アクセス制御の単位となるドメイン毎に用意する。暗号鍵 x によって語 k が暗号化された語を $k_{\mathrm{enc},x}$ とすると、$k_{\mathrm{enc},x} = hash(k,x)$ である。$hash()$ は一方向ハッシュ関数である。k と x を文字列として連結した後ハッシュする。

通常、Web サーバにおけるアクセス制御はディレクトリ単位で行われ、.htaccess 等のファイルに記述される。CSE における文書の公開・非公開の判断も.htaccess に基づいて行われる。CSE ではサーバ、ドメイン単位のアクセス制御を基本とする。ユーザ認証ディレクティブは同じサーバのみアクセス可とみなす。

Fig. 2 に CSE の更新時における Forward Knowledge の流れと語が暗号化される過程を示す。CSE では文書を公開する対象を同一サーバのみ、ドメイン内のみ、完全公開の 3 種類に分類し、同一サーバのみ、ドメイン内のみの場合は暗号鍵により暗号化する。同一サーバのみ公開の文書のための暗号鍵はそのサーバ外部には公開されない。ドメイン内のみ公開の文書のための暗号鍵はドメイン内のすべての LMSE と CS で共有されるが、ドメイン外には公開されない。

一方、Forward Knowledge を LS に送らずに、CS のドメイン内に閉じ込める方法も考えられる。しかし、この方式では検索時に多段階に CS を経由しなければならないため、遅くなってしまう。本提案方式は Forward Knowledge のサイズは増えるものの、検索は従来どおり高速である。

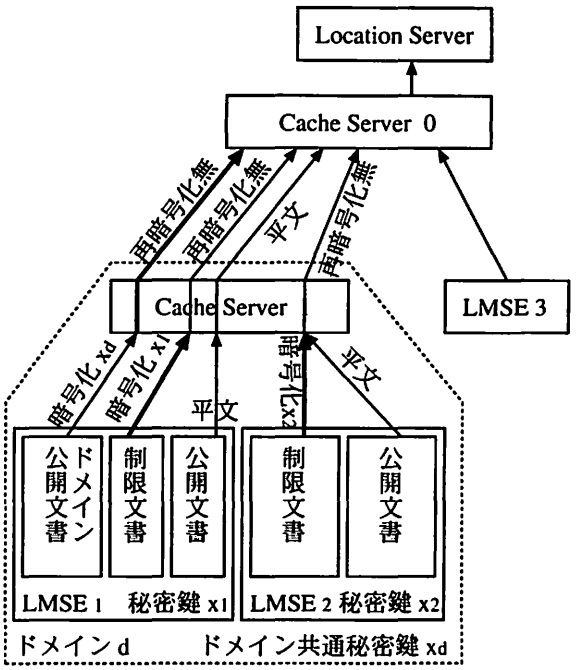

Fig. 2. 更新時の暗号化

更新時の手順は以下のようになる。

1. 各 LMSE は LSE を用いてインデックスを更新する。このとき、同一サーバのみ、同一ドメインのみ公開の文書に含まれるすべての語をそれぞれの暗号鍵を用いて暗号化する。インデックスには暗号化済の語が格納される。
2. LMSE は上位の CS にインデックスから取り出した Forward Knowledge を送信する。CS はより上位の CS か LS に近隣の LMSE から受け取った Forward Knowledge をまとめて送信する。CSE の規模など、場合によっては LMSE が LS に直接送信する。
3. LS はメタインデックスを更新する。

このように、非公開文書に含まれる語が平文のままネットワークを流れることは無い。また、LS のメタインデックス等にも暗号化された語が格納される。その代わりとして、LMSE が更新時に非公開文書のすべての語を暗号化する手間がかかり、更新時に送受信される Forward Knowledge のサイズが大きくなる。

メタインデックスは暗号鍵と一方向ハッシュ関数によって暗号化されており、復号化はできない。したがって、非公開文書を検索する時は、ユーザが与えた検索

語を暗号鍵と一方向ハッシュ関数で暗号化し、これを検索語として用いる。検索結果として得られる文書はアクセス制御されているのみで暗号化されていないため、普通に閲覧可能である。ハッシュ値が衝突しない限り、無関係のLMSEが検索対象サイトとしてCSに回答されてしまうことは起こらない。

図2において、$LMSE_1$のユーザが$LMSE_1$内のみ公開の文書と、ドメインd内部に対してのみ公開されている文書、完全に公開されている文書に含まれる語kを検索するとする。このときユーザが$LMSE_1$に対して与えるべき検索式は、

$$k \text{ OR } hash(k, x_d) \text{ OR } hash(k, x_1)$$

である。ここで、x_d、x_1はそれぞれdと$LMSE_1$の暗号鍵である。これをユーザが与えるのは繁雑であり、またユーザが暗号鍵を知っている必要があるので、ユーザはkのみを含む検索式をLMSEに与え、LMSEが与式のkをこの式に自動的に置換する。kと暗号化後のkはOR演算子で結合されているため、QbSSに悪影響を与えることは無い。また、検索の手順も、ユーザからクエリーを受け取ったLMSEがCSに検索要求を行う前にこの置換を行う必要があるだけで、従来と変わらない。

CSは自ドメイン内のLMSE、下位のCS認証のため、それらのリストを持つ。認証はIPアドレスベースで行う。このリストは管理者がメンテナンスする。

5 文書セキュリティの実装

CSE で用いている通信プロトコル (Cooerative Search Protocol、CSP) は、GMTP (Generic Message Transfer Protocol)[2] 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。本論文では、語の暗号化による文書セキュリティの実現のため、文献 [2][3][6] での定義を更新する。なお、GMTPの定義には変更はないが、CSがLMSEに秘密鍵を配布する時はSSLによって暗号化されたコネクション型通信路を用いる。便宜上、下記のような疑似RPC形式で表す。

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

以下にCS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。

GetSecKeyList : (SecKeyList) = `GetSecKeyList(void)`

LMSE または CS によってリクエストされる。`GetSecKeyList()` を受け付けた CS が持つドメイン暗号鍵のリストを返す。受け取った CS はリクエストを送ったクライアントの IP アドレスを確認し、CS の直接の子ノードであれば応答する。

以下にLMSEとCSが受け付けるメソッドを示す。

InvokeGetSecKeyList : (void) = `InvokeGetSecKeyList(URI)`

LMSE は CGI として実装されるため、暗号鍵の転送の必要が生じた場合に自発的に鍵を取得できない。CS は自発的に転送を開始できるため、上位の CS が下位の CS に鍵を直接送ってもよい。しかし、鍵を転送するメソッドを複数種類用意するのは意味が無いので、CS 同士の鍵の転送も `InvokeGetSecKeyList` を使って下位の CS に転送の開始を促す。`URI` には `GetSecKeyList()` を送るべき CS を指定する。

暗号鍵のリストは以下のようになる。ドメイン `Domain` は暗号鍵 `SecKey` に対応するドメインである。

```
SecKeyList = *(Domain SP SecKey CRLF)
```

ドメインは以下のように定義される。DNSのドメイン名（一部）かIPアドレスとマスクの組である。

```
Domain = (DomainName | (IPaddress SP Mask))
```

暗号鍵は任意桁の16進数のASCIIテキスト表現である。長さはLMSE、CSが自由に決めてよい。ただし、暗号鍵が長すぎるとハッシュ値が衝突する危険が大きくなる。

```
SecKey = *(HEXADIGIT)
```

`URI` はRFC2368で定義されるURIである。

6 評価

まず、Forward Knowledgeの暗号化にかかる時間について評価する。本論文で提案する方式ではLMSEが

更新時に Forward Knowledge として送る語を暗号化しなければならない。CSE の特長の一つに高速な更新があるが、暗号化に時間がかかれば更新時間が長くなってしまう。

Table 1 に一方向ハッシュアルゴリズムとして MD5 と SHA、公開鍵暗号として DSA を用いた場合に、語を暗号化するのにかかる時間を示す。語数は 135215 である。測定には Pentium4 1.8GHz の PC と PentiumIII 700MHz の PC を使用した。一方向ハッシュ関数では短時間に計算が終っている。CSE の更新時間は最短 1 分程度であるが、一方向ハッシュ関数は更新時間にほとんど影響を及ぼさない。公開鍵暗号は計算時間がかかりすぎるため適さないが、ハッシュの衝突が問題になるか更新時間がそれほど問題にならなければ使用は可能である。

次に転送する FK のサイズについて述べる。一般に、文書セットのサイズが大きくなっても、その文書セットに含まれる語の種類はそれほど増えない。この傾向は文書セットが大きくなるほど顕著である。Forward Kowledge のサイズは語の種類に比例するので、文書セットのサイズはそれほど影響しない。しかし、公開文書と非公開文書がある LMSE では、文書セットが二つあることになるので、転送すべき Forward Knowledge のサイズは増えてしまう。また、一方向ハッシュ関数の出力は MD5 の場合で 128 ビットの固定長で、CSE ではこれを 16 進数の ASCII テキスト表現で FK に用いる。したがって、FK の各語の暗号化後の長さは 32 byte になる。評価に用いた文書セットの場合、各語の平均長は 10.8 5byte であったので、語数が同じであれば暗号化後の Forward Knowledge のサイズは大きくなる。

Table 2 に各暗号化アルゴリズムの Forward Knowledge のサイズを示す。語を暗号化する前の Forward Knowledge のサイズは 1601499 byte であった。文書セットには暗号化速度の評価に用いたのと同じ文書セットを用いた。暗号化前、各暗号化方式ともに語の種類は全く同じで、概ね最悪値の評価といってよい。暗号化後には、平均して暗号化前の 3.37 倍になっているが、LAN 内での通信であることを考慮すれば、無視できるサイズといってよい。

7 まとめ

本論文では協調サーチエンジンにおいて、既に設定されている Web のアクセス権に基づいて文書毎に検索可・不可を設定する方式を提案した。この方式では検索不可にする文書の Forward Knowledge を暗号化してメタサーチサーバに送ることにより、暗号鍵と語を知らなければ検索できないようにする。暗号化済の語を総当たりで攻撃する方法はあるものの、現実的ではない。したがって、Web のセキュリティとしては十分実用になると考えられる。

Table 1. 暗号化時間の比較

暗号化方式		MD5	SHA	DSA
時間 [sec]	P4 1.8GHz	0.610	0.956	182
	P3 700MHz	1.34	1.64	326

Table 2. Forward Knowledge サイズの比較

暗号化方式	MD5	SHA	DSA
サイズ [byte]	5336224	6417776	4439529

参考文献

[1] 佐藤 永欣、上原 稔、酒井 義文、森 秀樹、"最新情報の検索のための分散型サーチエンジン"、情報処理学会論文誌、 第 43 巻、第 2 号、pp.321–331、情報処理学会 (2002)

[2] 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)

[3] 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"、DPSWS 2000, pp.205–210 (2000)

[4] 酒井義文、上原 稔、佐藤 永欣、森 秀樹、"協調サーチエンジンにおける検索クエリの最適な単調化"、DICOMO'2001, pp.453–458 (2001)

[5] 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"、DICOMO'2001, pp.465–470 (2001)

[6] 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大域的共有キャッシュ"、DPSWS 2001, pp.219–224 (2001)

[7] Nobuyoshi Sato, Minoru Uehara, Yoshifumi Sakai, Hideki Mori, "Persistent Cache in Cooperative Search Engine", In proc. of The 4th International Workshop on Multimedia Network Systems and Applications, pp.182–187 (2000)

[8] The MultiText Project, "MultiText", http://wheat.uwaterloo.ca/

[9] C. Weider, J. Fullton, S. Spero, "Architecture of the Whois++ Index Service", RFC1913, (1996)

[10] Darren R. Hardy, Michael F. Schwartz, Duane Wessels, "Harvest User's Manual", University of Colorado, Boulder, CU-CS-743-94, (1995)

[11] 日本電信電話 (株), "Ingrid: 広域情報検索システム", http://www. ingrid.org/index-j.htlm