最新情報の検索のための分散型サーチエンジン

佐藤永欣[†] 上原稔[†]
酒井義文[†] 森秀樹[†]

Web ページの情報検索ではサーチエンジンが必須である．一般的に使用されているサーチエンジンはロボットを用いて文書収集を行うが，集中型アーキテクチャであるため，更新間隔を短縮できないという問題点がある．そこで，我々は更新間隔を短縮するために分散型サーチエンジンである協調サーチエンジン（Cooperate Search Engine，CSE）を開発した．CSE では，各 Web サイトごとに配した局所サーチエンジンと，それを隠蔽する複数のメタサーチエンジンが協調して 1 つの大域サーチエンジンを構成する．CSE は Web サイトでインデックスを作成することで，更新間隔を大幅に短縮した．本論文では，CSE の構成と動作，その有効性について述べる．

A Distributed Search Engine for Fresh Information Retrieval

Nobuyoshi Sato,[†] Minoru Uehara,[†] Yoshifumi Sakai[†]
and Hideki Mori[†]

In Information Retrieval of Web pages, search engines are popular. Conventional search engines have the problem that is the update intervals are very long because they are based on centralized architecture, which collects documents by using robots. So, we have developed a distributed search engine, Cooperative Search Engine (CSE) in order to reduce the update intervals. In CSE, a global search engine is constructed with multiple Meta search engines, which hide each local search engine in each Web site. CSE can make the update interval very short by indexing on each Web site. In this paper, we describe the framework of CSE and its efficiency.

1. はじめに

インターネットが一般化してからすでに久しい．インターネットに関する話題を聞かない日はないほどインターネットは日常の中に融け込み，今日なお，その勢いを増している．

情報源という視点でインターネットをみると，巨大データベースと見なせるが，これは現在では主に WWW（World Wide Web）上の情報である．WWW 以前にも情報提供とそれらを検索する仕組みは用意されていたが，一般に普及するまでには至らなかった．

今日の WWW 上の情報検索ではサーチエンジンが必須である．WWW には体系的なインデックスがないため，検索が困難であった．まず，体系的なインデックスといえるディレクトリ型サーチエンジンが普及したが，現在ではロボット型サーチエンジンが主流である．

サーチエンジンでは，ロボットがリンクをたどって Web ページを巡回・収集する．一般的な集中型アーキテクチャのサーチエンジンとロボットの組合せでは，巡回に非常に長い時間が必要である．たとえば，早稲田大学の千里眼では日本全国の文書を収集するのに 1 週間が必要であった[11]．分散型ロボットが効率的に収集しても，極端な短縮化は難しい[11]．巡回がひととおり終わるまでの時間を更新間隔というが，これは Web ページに対する変更がサーチエンジンに反映されるまでの時間のうち，最悪の場合である．

インターネットの企業等の通信手段としての利用は一般的であるが，更新間隔が長いため，インターネット上の情報が企業活動に活用されることは多くない．

しかし，その一方，組織内での情報の活用の必要性は非常に高まっている．データマイニング（Data Mining）やナレッジマネジメント（Knowledge Management）が情報の活用として行われるようになり，情報を活用・共有することで企業の生産性を上げることが当然とされるようになったためである．

組織内の情報活用・共有の手段として，サーチエンジンは非常に有効である．多くの商用サーチエンジン

† 東洋大学工学部情報工学科
Department of Information and Computer Sciences, Faculty of Engineering, Toyo University

の運営会社が企業向けにサーチエンジンを供給している．また，Namazu[1] や SGSE[2] に代表されるような，GPL（GNU General Public License）等のフリーなライセンスに基づいて配布される小規模サイト向けサーチエンジンも一般的である．しかし，これらは集中型アーキテクチャであるため，更新間隔の大幅な短縮は困難である．

イントラネットの Web サーバはインターネットに比べ非常に少ないが，イントラネットのサーチエンジンに要求される更新間隔はきわめて短い．したがって，イントラネットでは相対的に更新間隔が問題となる．

本論文で提案する協調サーチエンジン（Cooperative Search Engine，CSE）[15),18),20)] は，更新間隔を短縮することを目的に開発された分散型サーチエンジンである．各 Web サーバに，その Web サーバの文書だけを検索する局所サーチエンジンを用意し，これらをメタサーチエンジンで統合することにより，1 つの大域サーチエンジンとして動作する．これにより，検索時の速度は犠牲になるが，更新時の通信を減らし，更新間隔を大幅に短縮することが可能である．本論文では，CSE のしくみと有効性について述べる．

本論文の構成は以下のとおりである．2 章では分散型情報検索の関連研究，3 章では CSE の概要と動作について述べる．4 章では CSE の評価について述べる．最後にまとめを述べる．

2. 関 連 研 究

まず，更新間隔を短縮する研究について述べる．いくつかのアプローチがあるが，分散型ロボットを用いる，一度に大量に収集する，並列度を増す，収集方法を工夫するといったアプローチがある．

分散型ロボットとしては，Harvest[5] の Gatherer，PRSM[11] 等がある．

PRSM は日本中の Web ページを 24 時間で収集することを目指した分散協調型 Web ロボットである．ロボットと収集対象 Web サーバとのネットワーク的距離を考慮して，自動的に最適な収集対象を割り当てる．7 カ所のロボットで 103 サーバの文書収集を行い，約 5〜20 倍の性能を達成した．これは文書へのアクセス手段に HTTP を用いた例としては現在最善のものと思われるが，裏を返せば HTTP を用いる限り限界があることを示している．

我々が CSE 用に開発した Web ロボット[3] は，一度のアクセスで複数のページを取得するアーカイブ・アクセス（後述）を用いることで，この限界を超えようとしている．

ロボットを分散するよりも並列度を高めるというアプローチもある．文献 13) によると，80000 もの超並列で処理しても 10 Mbps の帯域を使い切ることはない．この手法はサイト数が多く分散しているインターネットでは有効と考えられるが，サイト数の少ないイントラネットには適用できない．

ロボットの動作はロボット排除規則[7] で制限される．この規則は，同一サイトから収集する最短時間間隔を 5 分以上保つよう推奨している．この間隔を厳守した場合，1 日に取得できるページ数は 1 サイトあたりたかだか 288 ページで，実用に耐えない．したがって，多くのロボットは少なくとも時間間隔に関して，ロボット排除規則を無視している．また，一般的に，ロボットによる収集では通信にともなうオーバヘッドが大きく，極端に更新間隔を短縮することは困難である．

収集方法を工夫して更新時間を短縮しているのは Infoseek[6]，FreshEye[25] 等である．Infoseek は，全サーバ一覧，サーバ上の全ファイル一覧，更新ファイル一覧等を用いて収集を効率化しているが，これにはサイト管理者の協力が必要である．また，CSE と同様のメタインデックスを用いた分散検索方式も提案している．電子指紋を利用している点に特徴がある．

FreshEye は，集中型アーキテクチャで 10 分間隔の更新を実現している．これは収集するドメインを特定し，さらに更新頻度の高い順に収集することで実現している．しかし，全ドメインを収集対象としていないことが問題であり，大学等学術系の機関は収集対象となっていないことがある．

これらのアプローチでは原理的にすべての文書を更新対象にするのは著しく困難である．また，イントラネットに要求される更新間隔を実現するのも困難である．

一般的なメタサーチエンジンには，利用するサーチエンジンの種類によって語の重み付けが異なるため，同じ文書であっても順位が異なるという問題がある．NEC の Inquirus[9] は，メタサーチの結果を用いて，文書をダウンロードしてスコアを再計算することでランキングの矛盾を解消している．しかし，この方法ではランキング計算に時間がかかる．CSE では Namazu の重み付け方式に統一することによって，この矛盾を解消している．しかし，Namazu 以外のサーチエンジンのインデックスを再作成する必要がある．

次に，分散検索に関する研究について述べる．分散検索の研究は従来から多く行われており，WWW の検索以外では，人を探す Whois++[22] 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 [5]，Ingrid [19] 等があげられる．

Whois++では Forward Knowledge（FK）を centroid という単位でまとめる．各サーバは FK を基にクエリを転送し，自分が知らない情報の検索を依頼する．これをクエリルーティングという．

Harvest は Gatherer と Broker のコンポーネントからなる．Gatherer は文書を収集し，要約を SOIF（Summary Object Interchange Format）という形式で Broker に転送する．SOIF はオブジェクトの属性や要約を共通的に利用するための形式で，著者名，タイトル，キーワード 等の要約を含む．Harvest では，全文検索を行う場合には SOIF に全文を含む必要があり，Gatherer は収集した文書に属性情報を付加して Broker に転送している．インデックスは Broker で作成される．また，Broker は検索要求を受け付け，他の Broker と連携して分散検索を行う．Harvest では，Gatherer 自体は HTTP による文書収集以外にファイルシステムを通してファイルに直接アクセスすることによる収集もできるが，Gatherer がインデックス生成を行わないため Broker へ文書全体を転送する必要がある．このため，CSE ほどの更新間隔の短縮は難しい．

Ingrid はトピックレベルの検索・閲覧を目指した情報インフラストラクチャである．Ingrid では，収集したリソースにリンクを張り，独自のトポロジを構築する．このトポロジを Forward Information（FI）サーバが管理し，Ingrid ナビゲータが FI サーバと通信しながら経路探索することで検索を行う．Ingrid は柔軟であるが，経路を逐次的に探索するため検索時の遅延が大きくなる可能性がある．CSE は経路探索を集中して行うため，ボトルネックが生じる危険があるが，検索時の遅延は小さくて済む．

また，分散検索では文書のランキングも難しい．Web の全文検索では，指定した語を含む全ページを検索結果として表示する．しかし，該当するページが多いと重要なページが埋没してしまうため，スコアによるランキングが必要となる．また，検索結果 1 ページに表示される件数を 10〜20 件に絞り，ユーザの要求に応じて次々とページ単位で表示する方式を継続検索という．継続検索は多くのサーチエンジンが採用している．継続検索では，ユーザは最初の 1〜2 ページしか見ない傾向があるため [10]，ランキングの正しさはサーチエンジンの使いやすさに直結する．

代表的なランキング方式には，古典的であるが $tf \cdot idf$ 法がある [4]．ここで，tf（Term Frequency）は検索語が文書中に出現する頻度であり，idf（Inverse Document Frequency）は検索語の珍しさを表し，$\log(N/n)$ である．また，N は検索対象とする文書の総数，n はその検索語を含む文書数である．文書のスコアを tf と idf の積で表すランキング方式が $tf \cdot idf$ 法である．

分散検索では，idf を局所的に決定できないため，2 パスの通信を行う必要があり [8]，通信遅延が大きくなってしまう．CSE は $tf \cdot idf$ 法を採用するが，更新時に idf の計算に必要な情報を一緒に収集し，検索時に idf を計算・配布することにより通信を 1 パスで済ませている．

3. 協調サーチエンジン

更新間隔の短縮化を主な目的として，我々は協調サーチエンジン（Cooperative Search Engine，CSE）を開発した．ここでは，CSE の構造と動作について述べる．

3.1 CSE の概要

まず，CSE の基本的な原理を説明する．CSE では更新間隔を短縮するため，局所的なインデックスを作成し，このサマリ情報を全体で一括管理する．すなわち，だれ（サイト）が何（語）を知っているかという情報を管理者に送り，一括管理する．この情報を Forward Knowledge（FK）といい，だれが何を知っているかを表すメタ知識で，Ingrid の FI と同じものである．検索時には，まず管理者にだれがその語を知っているかを問い合わせ，その語を知っているサイトにだけ検索を依頼する．このように，実行時に互いに通信して協調しながら検索するため，集中型アーキテクチャより検索時の応答が遅いという問題がある．しかし，そのかわりとして更新間隔を極端に短縮することが可能である．そこで，いかに検索時の通信を抑え，応答を早くするかが CSE の課題となる．

CSE は**図 1** に示されるような以下のコンポーネントから構成される．

- Location Server（LS）：LS は FK を一元管理するサーバである．CSE 全体で 1 つだけ存在する．LS は FK を元に文書への経路（文書を所有する LMSE）を検索する．
- 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 とも考えられるが，検索結果をキャッシュしない．CS は検索結果をキャッシュすることで高速な継続検索を実現する．継続検索とは検索結果すべてを一度に表示せずに区切って表示する方

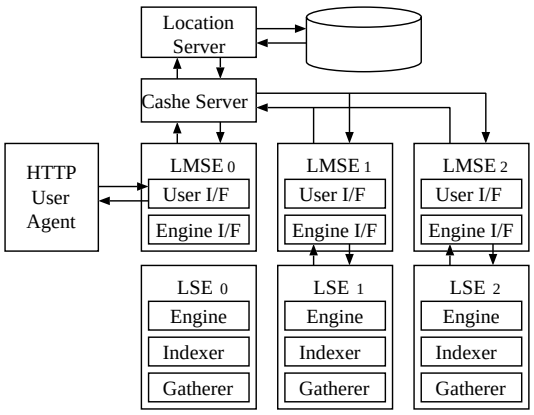

図 1 CSE の概要
Fig. 1 The components and structure of CSE.

法のことである．詳しくは後述する．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 に転送する（図 1 の User I/F）．また，後述の LSE を用いて局所的な検索を行う（図 1 の Engine I/F）．LMSE は LSE をラッピングして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 1 の Indexer），検索（図 1 の Engine）を行う．CSE では Namazu や SGSE を LSE として使用できる．並列検索やフレーズ検索等高度な検索を提供する独自の Namazu 用インデクサも開発している[24]．

Namazu は小規模サイト向けの日本語全文検索エンジンである．Namazu は情報処理学会の Web サイト[14]，オープンソースコミュニティ等をはじめ，Web サイトの検索サービスとしてすでに広く普及している．CSE ではすでに使用されている Namazu を何も変更せずに利用できるため，導入と移行が容易である．

3.2 CSE の更新時の動作

まずはじめに，インデックス更新手順の用語を定義する．インデックス更新は，以下の順序で行われる．

(1) 文書収集：Web サーバから文書を収集する．
(2) インデックス作成：インデックスを作成する．
(3) インデックス転送：作成されたインデックスをサーチエンジンに転送する．同一サイトでインデックス作成と検索を行う集中型サーチエンジンでは不要なことがある．

インデックス更新はパイプライン処理することも可能である．また，文書収集とインデックス作成は並列処理で行うことが可能であり，パイプライン処理と組み合わせることでインデックス更新時間を短縮できる．一般に文書収集に最も時間がかかり，次いでインデックス作成，インデックス転送の順であるが，これは取り扱う情報量の問題である．したがって，高速に更新を行うためには情報量を極力小さくして取り扱い，大きな情報量を取り扱う場合は極力高速な方法を用いる必要がある．

次に，文書収集の方式を分類し，用語を定める．文書収集の方式には以下の 3 方式がある．

直接アクセス NFS やローカルの記憶装置等，ファイルシステムを通して直接ファイルをアクセスする方式．

アーカイブ・アクセス 複数ファイルをアーカイブして転送する検査 CGI を用いることにより，1 度のアクセスで複数の文書を転送する方式．

HTTP アクセス HTTP 経由で 1 度のアクセスで 1 つずつ文書を転送する方式．一般的に用いられる．

これら 3 方式のうち直接アクセスが最も高速で，次いでアーカイブ・アクセス，HTTP アクセスの順であるが，これは収集対象とのネットワーク的な距離や，通信遅延と通信量の問題である．

次に CSE の更新時の動作について述べる．CSE の更新時のアルゴリズムは以下のとおりである．

(1) LSE の Gatherer が対象サイトの文書（Web ページ）を収集する．このとき，できる限り高速な方法で収集する．すなわち，直接アクセスが可能なら直接アクセスを用い，アーカイブ・アクセスが可能ならアーカイブ・アクセスを用いる．これらが不可能な場合だけ HTTP アクセスを用いる．
(2) LSE の Indexer がインデックスを作成する．
  (a) $LMSE_i$ の Engine I/F は対応する LSE から全文書数 N_i と，語の集合 K_i，すべての語 $k \in K_i$ に対して k を含む文書数 $n_{k,i}$ を求め，自身の URL とともに LS に送信する．
  (b) LS は各 $LMSE_i$ から送られた $n_{k,i}$ と N_i を記録する．

CSE ではこのように極力高速な方法を用い，情報量を小さくして保管し，コンポーネント間での通信に用いる．他の分散サーチエンジンと比較すると速度の点で有利である．また，CSE ではボトムアップで更新を行うが，いっせいに更新を行うことも可能である．この場合は LS が LSE の Gatherer を起動するように

各 LMSE に指令を出す．

なお，各モジュール間で送受されるデータは主として $tf \cdot idf$ 法に基づくスコアを分散計算するために用いられる．この方式を分散 $tf \cdot idf$ 法と呼んでいる[17]．分散 $tf \cdot idf$ 法ではスコアは検索時でなければ最終的に決定できない．したがって，詳細な説明は 3.3 節で述べることにする．

CSE では前述の 3 つの文書収集方式をサポートしている．そのため，Web サーバの状態を検査する CGI を用意し，アーカイブ・アクセスを実現している．この検査 CGI は，Web サーバの SERVERROOT とユーザの一覧から，SERVERROOT とユーザ単位のアーカイブを作成し，転送する．

3.3 CSE の検索時の動作

ここでは CSE の検索時の動作について述べる．CSE は更新時の性能の向上を目的として，検索時の性能を犠牲にしている．以下は CSE の検索時の動作の概要である．

(1) ユーザからの検索要求を受け付けた $LMSE_0$ は問合せ内容を CS に送る．
(2) CS は LS に検索要求を満たす文書を持つ LMSE の集合を問い合わせる．
(3) CS はその結果を用いて，各 LMSE に検索要求を送信する．
(4) LMSE は LSE を用いて検索し，その結果を CS に返送する．
(5) CS はこれらの検索結果をまとめて $LMSE_0$ に返送する．
(6) $LMSE_0$ はユーザに検索結果を表示する．

このように，CSE では検索時に多くの通信が発生するため，検索の高速化のためには検索の手順そのものを工夫して文書にたどり着くまでのパス数を減らすとともに，通信量と通信遅延の削減が重要である．すでに述べたように，CSE では文書への経路を LS が FK を元に集中して検索するため，文書にたどり着くまでのパス数は他の分散サーチエンジンより小さい．CSE では以下のような方法を用いて通信量と通信遅延を削減している．

CSE では論理式に基づくブーリアン検索が可能である．CSE のブーリアン検索では，and，or，not 演算子を使用できる．ここで，not は情報検索の分野で一般的な差分を表す二項演算であって，a not b は，語 a を含むが b を含まないページを表す．クエリ A，B の検索対象サイトを S_A，S_B とすると，

- A and B の検索対象サイトは $S_A \cap S_B$．
- A or B の検索対象サイトは $S_A \cup S_B$．
- A not B の検索対象サイトは S_A．

となる．この絞り込みにより検索時の通信量を削減することができる[21]．

また，一般的なサーチエンジンは検索結果を，たとえば 10 件ごとのページに分割し，ユーザの要求に応じて各ページを次々に表示する．これを継続検索[8]という．CSE では継続検索時の通信量と通信遅延を最小化するため，バックグラウンドで検索結果を 1 ページ分ずつ先読みし，キャッシュしている．これは多くのユーザは検索結果の最初の 1〜2 ページしか必要としていないことが経験的に知られているため[10]，有効な方法である．

さらに，継続検索時においては，各サイトが検索結果とともに次回の継続検索で得られる最大スコアを回答する．これに基づいて先読み対象サイトを限定し，継続検索時の通信量を最小限に抑えている[16]．N_p 件の検索結果を含むページを 1 ページ表示する際に検索する必要のあるサイトはたかだか N_p サイトである．しかし，CSE では実際に LMSE で検索するまで各文書のスコアを知ることができない．このため，少なくとも 1 ページ目の検索では CS は適合する文書を持つ可能性のあるサイトすべてに検索要求を送る必要がある．しかし，2 ページ目以降の検索においては，最初の問合せの際に次回の検索で得られる文書の最大スコアを LMSE が回答することにより，CS はどのサイトが次のページの表示に必要な検索結果を返すかを知ることができる．これを用いて，先読み対象サイトを N_p サイトに限定することが可能である．本論文では，このような方法をスコアに基づくサイト選択と呼ぶ．

分散 $tf \cdot idf$ 法は $tf \cdot idf$ 法によるスコア計算を分散サーチエンジンに適用したスコア計算法である．分散 $tf \cdot idf$ 法の計算式を以下に示す．

$$score(d,k) = tf(d,k) \cdot idf(k)$$
$$idf(k) = \log \frac{\sum_i N_i}{\sum_i n_{k,i}}$$

ここで，$score(d,k)$ は文書 d の語 k に関するスコア，$tf(d,k)$ は k が d に出現する頻度，N_i は $LMSE_i$ の文書数，$n_{k,i}$ は $LMSE_i$ において k を含む文書数である．

CSE では，更新時に各 LMSE から N_i，$n_{k,i}$ を収集し，LS が管理する．検索時には LS が $idf(k)$ を計算し，CS からの検索対象サイト問合せに回答するときにまとめて配布する．配布された $idf(k)$ は CS が各 $LMSE_i$ に検索要求を送る際にまとめて配布され，$tf \cdot idf$ 法によるスコア計算に用いられる．したがって，CSE ではスコア計算に分散 $tf \cdot idf$ 法を用いるこ

とによる余分な通信は発生しない．

CSE の検索は CS が主体となって行う．CS は検索式 E ごとに用意したソート済みキュー Q_E と，E と $LMSE_i$ ごとに用意した未ソートキュー $Q_{E,i}$ を持ち，これに文書の URL とスコアからなる検索結果を格納する．CSE の検索時のアルゴリズムは以下のとおりである．

(1) ユーザは $LMSE_0$ に E，表示開始件数 N_s，ページ単位件数 N_p を送信する．

(2) $LMSE_0$ は E，N_s，N_p を CS に送信する．

(3) CS は Q_E があるかどうかをを調べ，なければ新規検索であるため E を LS に転送する．Q_E がある場合は，LS に問い合わせずに (7) を実行する．

(4) LS は E 中の語を含む LMSE を検索する．E に基づく検索対象サイト絞込みを行い，検索対象サイト集合 S_E を求める．

(5) LS は E に含まれるすべての語 k に対する $idf(k)$ を計算する．その後，S_E と，全文書数 N，$idf(k)$ を返送する．

(6) CS は LS から返送されたデータをキャッシュし，Q_E と $Q_{E,i}$ を用意する．

(7) CS は $N_s + N_p - 1 \leq |Q_E|$ のとき，Q_E の N_s からの N_p 件を $LMSE_0$ に返す．ここで，$|Q_E|$ は Q_E の長さである．

(8) CS は，$|Q_E| < N_s + 2N_p - 1$ のとき，検索要求を送る $LMSE_{E,i}$ を以下のように決定する．
  (a) $LMSE_{E,i}$ が次回の検索要求で返す検索結果の先頭のスコア $s_{E,i,next}$ がない場合は，LS が E に適合する文書を持つと回答した LMSE すべて，すなわち S_E を $LMSE_{E,i}$ とする．
  (b) $s_{E,i,next}$ がある場合は，$s_{E,i,next}$ を降順にソートし，$s_{E,i,next}$ が大きい LMSE から順に N_p 個の LMSE を $LMSE_{E,i}$ とする．

(9) CS は $\forall LMSE_{E,i} \in S$ に対して以下を行う．$N_s + 2N_p - 1 - |Q_E| > |Q_{E,i}|$ ならば，$LMSE_{E,i}$ に E と各 $idf(k)$ を送信し，$T_i + 1$ 件目からの $N_s + 2N_p - 1 - |Q_E| - |Q_{E,i}|$ 件を要求する．ここで，T_i は $LMSE_i$ から返された結果の個数で，$Q_{E,i}$ へ格納した検索結果の総数，$|Q_{E,i}|$ は $Q_{E,i}$ の長さである．

(10) $LMSE_{E,i}$ は E に含まれる語 k に対して $LSE_{E,i}$ に検索を依頼し，k を含む tf の降順の文書リスト $D_{k,i}$ を求める．さらに E に基づく論理演算を行うと同時に文書 d のスコア $w(d, E)$ を求める．なお，$w(d, E)$ は以下の式で決まる．
  (a) $w(d, k) = tf(d, k) \cdot idf(k)$
  (b) $w(d, A \text{ and } B) = \min(w(d, A), w(d, B))$
  (c) $w(d, A \text{ or } B) = \max(w(d, A), w(d, B))$
  (d) $w(d, A \text{ not } B) = w(d, A)$
  ここで，k は単独の語，A と B は任意の論理式である．最後に，$w(d, E)$ 順の文書リスト $D_{E,i}$ を求め，これを検索結果とする．

(11) $LMSE_{E,i}$ は検索結果をスコアの降順にソートして CS に返送する．このとき，E に関する次回の検索で $D_{E,i}$ の続きとして得られる検索結果のスコアで最大のものを $s_{E,i,next}$ として同時に返送する．

(12) CS は $\forall LMSE_{E,i} \in S_E$ の検索結果を $Q_{E,i}$ に追加する．

(13) CS は $Q_{E,i}$ のどれか 1 つが空になるまで，すべての $Q_{E,i}$ の中から最大のスコアを持つ結果を取り出し，Q_E へ格納する（マージソート）．このとき，CS がまだ検索結果を返送していない場合は，この途中で $N_s + N_p - 1 \leq |Q_E|$ の条件を満たしたとき，Q_E の N_s からの N_p 件を検索結果として $LMSE_0$ に返す．

(14) $LMSE_0$ は検索結果を HTML 形式に整形してユーザに返送する．

ここで，(9) から (11) までは並列に実行される．

4. 評　　価

4.1 更新性能の評価

インデックス更新には以下の 2 つの種類がある．

- 単純更新：新規ファイル，または変更されたファイルの情報を追加的に更新する方法．
- 完全更新：インデックスを完全に作り直す方法．削除されたファイルの情報を抹消することができる．

完全更新はインデックスをすべて削除してから，単純更新することと同等である．このため，単純更新より時間がかかる．単純更新を日常的に行い，完全更新はメンテナンスとして実行することが多いため，CSE では単純更新時間の短縮を重視している．

CSE では LSE が更新作業の大部分を行うが，多くの時間がかかる．そこで，文書収集，インデックス作成を並列・分散処理する LSE を開発した[23]．この LSE の有効性を確認するため，まず，東洋大学内最大のサイトである工学部メディア・コンピューテングセンター

表 1 予備更新実験の評価
Table 1 The evaluation of preliminary examination.

	文書収集 [h : mm : ss]	インデックス作成 [h : mm : ss]	インデックス転送 [h : mm : ss]	計 [h : mm : ss]
Namazu 完全	0 : 25 : 50	0 : 20 : 32	0 : 00 : 00	0 : 46 : 22
Namazu	0 : 19 : 51	0 : 01 : 27	0 : 00 : 00	0 : 21 : 18
CSE	0 : 00 : 09	0 : 01 : 27	0 : 00 : 18	0 : 01 : 54
並列 CSE	0 : 00 : 02	0 : 00 : 37	0 : 00 : 11	0 : 00 : 50

の Web サーバを対象とした更新実験を行った．次に東洋大学の文系最大サイトで，LMSE を導入できない ToyoNet の Web サーバを対象とした実験を行った．

予備実験として PC に，メディア・コンピューテングセンターの約 2,000 人分の Web ページ（8,000 ファイル，総バイト数 12 MB）を移設して行った．実験に用いた PC は Celeron 333 MHz，メモリ 128 MB と Celeron 400 MHz（デュアルプロセッサ），メモリ 128 MB の 2 台で，それぞれ FreeBSD がインストールされている．並列処理はこれら 2 台の PC を用いたが，デュアルプロセッサ機がファイルサーバを兼ねた．この結果を**表 1** に示す．ここで，比較対象として，wget と Namazu を用いた完全更新の場合（Namazu 完全）と同単純更新の場合（Namazu），CSE の単純更新で並列処理を行わない場合（CSE）と並列処理を行う場合（並列 CSE）の 4 通りを行った．また，最短時間を測定するため単純更新の測定は更新ファイル数 0 個で測定した．単純更新では更新されるファイルはほとんどないため，実際にインデックスを作成する処理も短時間である．しかし，インデクサは更新されたファイルを見つける必要があり，この処理が大部分を占める．

これにより以下のことがいえる．まず，単純更新方式は完全更新方式に比べて，インデックス作成時間を大幅に短縮する．次に，直接アクセスは HTTP アクセスに比べて文書収集時間を大幅に短縮する．また，並列処理は並列度が低いときにはインデックス作成時間を約 $1/N$ にする．これは，文書のわかち書き化，単語出現回数の測定等のインデックス作成の計算そのものはほとんど I/O をともなわず，主に CPU の処理能力に依存するため台数効果がそのまま出るが，並列度が上がると，後で述べるようにインデックス作成の計算そのもの以外の文書ファイルの読み込み，インデックスの書き出しによる I/O がボトルネックとなるためである．

次に本実験として，メディア・コンピューテングセンターの WS（Sun SPARCstation 20，SuperSPARC Ⅱ 75 MHz デュアルプロセッサ，メモリ 224 MB）を用いて評価実験を行った（**図 2** 参照）．CPU の性能差

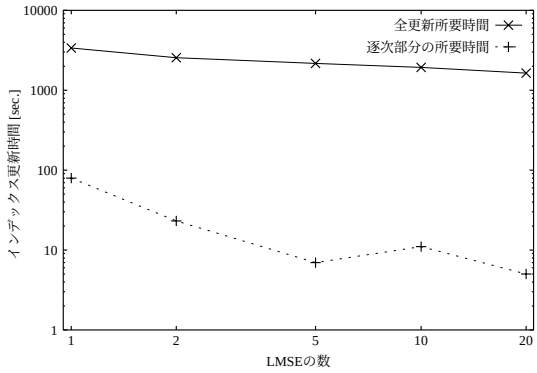

図 2 本更新実験の評価
Fig. 2 The evaluation of main examination.

により予備実験を下回った．また，5 台以上では十分な台数効果が得られなかったが，これは通信路に用いる NFS によるファイル共有が原因であった．インデクサによる文書ファイルの探索・読み込みのアクセスが NFS サーバに集中し，HDD がつねにシークを続ける状態になり，I/O 待ち時間が大幅に増加する．また，10 台で gcc を同時にコンパイルしながらインデックスを作成したところ，無負荷時の 1.4 倍の時間で終了した．通常，このような負荷がすべての計算機につねにかかることはないので，通常の負荷変動を考慮してもメディア・コンピューテングセンターの全文書を 30 分〜1 時間以内で更新できると考えられる．予備実験の PC を用いればさらに短縮できることが分かっているため，今後は検索専用サーバの開発を行う．

次に LMSE の計算機の処理能力の差，文書量の差等によるインデックス作成時間のばらつきをなくすため，他のアイドル状態の LMSE にインデックス作成を依頼することで負荷分散する実験を行った．A，B，C の 3 台の LMSE に文書 1,400 個ずつをインデックスし，それぞれ 200 個，約 1 MB の文書を追加する単純更新を行った．A，B がほぼ同等の性能，C が A，B の半分弱である．この結果を**表 2** に示す．この結果，LMSE の性能差，文書量の差によるインデックス作成時間のばらつきをなくすことができた．

次に ToyoNet を対象としたアーカイブ・アクセスの評価を行う．ToyoNet は商用インターネットサービス

プロバイダにアウトソーシングされているため，ユーザが使用可能なディスク容量が厳しく制限されていて，LMSE 導入はできず，直接アクセス法も使えない．したがって，他の計算機でインデックスを生成，管理しなければならない．ToyoNet では CGI は使用可能なため，文書収集にアーカイブ・アクセスを用いる．**表 3** に HTTP アクセスとアーカイブ・アクセスの文書収集時間を示す．ここで，HTTP アクセスは wget を用い，アーカイブ・アクセスでは複数ユーザのファイルを CGI でアーカイブしてまとめて転送した．このとき転送したファイルは圧縮前で合計約 80 MB で，圧縮することでほぼ 1/3 強のサイズになる．アーカイブ・アクセスを用いることで，文書収集時間を HTTP アクセスと比べて 1/3 以下にできることが確認できた．これは，圧縮して転送することによる転送時間の短縮のほかに，HTTP で文書を 1 つずつ取得する際のオーバベッドを避けることができたためと考えられる．**表 4** に 50 ユーザごとにアーカイブ・アクセスを行った場合の ToyoNet の更新時間を示す．実験条件は ToyoNet を対象とした以外は予備実験とまったく同じである．

4.2 検索性能の評価

CSE では，更新時間短縮の代償として，検索時に通信による遅延が生じてしまう．**表 5** に局所検索の応答時間を示す．SGSE は Perl で記述されているため C で記述された Namazu より遅い．CSE は通信を必要とするため SGSE よりもさらに遅い．

CSE では，このオーバヘッドを最小化するため，並列検索を行っている．**表 6** に 24 サイトを Namazu を用いて逐次検索した場合と，CSE で並列に検索した場合の応答時間を示す．CSE の AND と OR の所要時間の差は結果の件数の差による通信時間の違いが原因である．次に，CSE の並列検索のスケーラビリティを**図 3** に示す．台数 N に対して $N \leq 10$ までは $O(\log N)$，$N > 10$ では $O(N)$ であることが分かる．また，LMSE が 10 のときに LMSE との通信による輻輳が始まり，20 では完全に輻輳していた．$N > 10$ で $O(N)$ となるのは，この輻輳が原因と考えられる．したがって，検索対象を絞り込んで不要な検索を行わないこと，継続検索の効率化が重要となる．また，図 3 の実験中，並列，逐次部分の実行時間を調べたところ，並列処理部分が検索所要時間のほぼ全体を占め，逐次処理部分は最長でも 0.065 秒にすぎなかった．一般に，ユーザがサーチエンジンの検索結果を待つ時間は 8 秒が限度といわれている．LMSE が 10 以上では $O(N)$ で検索所要時間が増えることを考えると，LMSE が 50 程度で 8 秒を超えると考えられる．

我々は文献 21) において検索対象サイトの絞り込みについて議論し，理論的には検索対象サイト数を約 40%に削減できると予想した．学内 27 サイトを対象とした評価実験では，LS に登録されているすべての語 k に対して，k を含む LMSE 集合 S_k の要素数 $|S_k|$ は平均 1.6 となり，サイト数を 6%にまで削減できる

表 2 他の LMSE への処理依頼による負荷分散
Table 2 Load balancing by requesting to other LMSE.

	A[m : ss]	B[m : ss]	C[m : ss]
処理依頼あり	2 : 33	3 : 22	3 : 59
処理依頼なし	2 : 20	3 : 22	6 : 29

表 3 HTTP アクセスとアーカイブ・アクセスの文書収集時間
Table 3 Document collecting times HTTP access vs. archived access.

アクセス方式	ユーザ数	文書収集 [h : mm : ss]
HTTP		1 : 35 : 32
アーカイブ	1	0 : 52 : 17
	10	0 : 28 : 01
	20	0 : 26 : 57
	50	0 : 24 : 10
	100	0 : 24 : 19

表 5 1 語における局所的検索の応答時間の比較
Table 5 Local retrieval times in a word.

	Namazu [sec]	SGSE [sec]	CSE [sec]
応答時間	0.046	0.448	1.267

表 6 2 語における逐次検索と並列検索の応答時間の比較
Table 6 Sequential vs. parallel retrieval times in two words.

	Namazu（逐次）[sec]	CSE（並列）[sec]
AND 検索	5.6	1.5
OR 検索	5.6	5.0

表 4 アーカイブ・アクセスの更新時間
Table 4 Update interval times in archived access.

	文書収集 [h : mm : ss]	インデックス作成 [h : mm : ss]	インデックス転送 [h : mm : ss]	計 [h : mm : ss]
Namazu 完全	1 : 35 : 32	1 : 16 : 25	0 : 00 : 00	2 : 51 : 57
Namazu	1 : 35 : 32	0 : 01 : 03	0 : 00 : 00	1 : 36 : 35
CSE	0 : 24 : 10	0 : 01 : 03	0 : 01 : 07	0 : 26 : 20
CSE 並列	0 : 06 : 51	0 : 00 : 23	0 : 00 : 34	0 : 07 : 48

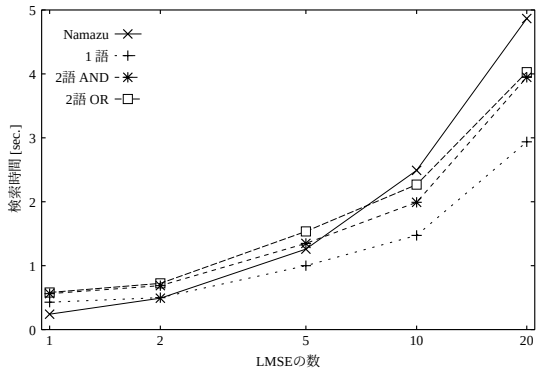

図 3 CSE のスケーラビリティ
Fig. 3 The scalability of CSE.

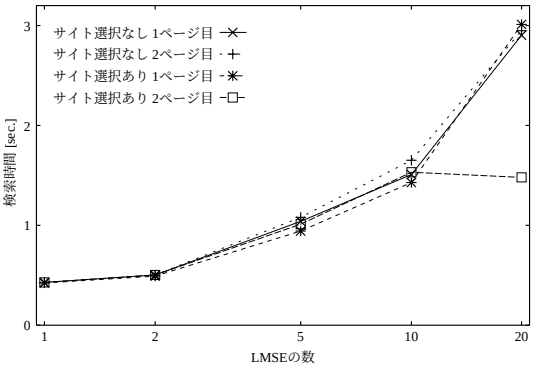

図 4 スコアに基づくサイト選択の評価
Fig. 4 The evaluation of score based site selection.

表 7 CS の性能評価
Table 7 The performance evaluation of CS.

ヒット件数	CS なし [sec]	CS あり [sec]	
		1 ページ目	2 ページ以降
903	2.97	0.80	0.31
544	1.75	0.66	0.32

ことが分かった．これは学内では一部のサイトに文書が偏っているためである．検索対象サイトの絞り込みを用いることによって，LMSE 数が 100 程度の CSE を構築しても検索は数秒で終了し，実用に耐えると考えられる．

次に継続検索の評価について述べる．CSE では 1 ページに 10 件の検索結果を表示し，継続検索を行っている．東洋大学のある WWW プロキシサーバのログを解析した結果，継続検索は全検索の 70%に及ぶことが判明した．そこで，継続検索の効率化のために CS を導入した[12]．**表 7** に CS の性能評価を示す．このとき，語の重要度は中程度の頻度が最も高いため，ログの中から中程度の頻度の語を選択して検索を行った．CS がない場合は毎回全件数を検索する必要があるが，CS がある場合は必要な件数のみ検索するだけで済む．この結果，1 ページ目で約 25〜40%，2 ページ以降では約 10〜20%まで検索時間を削減できた．特に 2 ページ以降の検索は SGSE より高速であった．以上のことから CSE の検索時間は集中型サーチエンジンと遜色なく，CSE は実用に耐えうると考えられる．

継続検索時の 2 ページ目以降の検索をさらに効率化するため，CS にスコアに基づくサイト選択をとり入れ[16]，CSE の検索時の動作は前章で述べたアルゴリズムとなった．2 ページ目以降の検索は CS がバックグラウンドで実行する．したがって，見かけ上，スコアに基づくサイト選択の効果が表に出ることはほとんどないが，CS が継続検索に費やす時間を短縮したり，継続検索時の通信量を削減したりできる．また，LMSE の数が増加したときにはスコアに基づくサイト選択は必須である．**図 4** にこの評価を示す．各 LMSE のインデックスはまったく同じものを用い，検索語には全文書に現れる単語を用いたが，これは最悪の場合である．サイト選択をした場合の 2 ページ目では，LMSE の数が 20 であっても LMSE が 10 のときとほぼ同じ時間で検索を終わっている．LMSE の数が増えても，検索時間の大幅な増加はないと考えられる．

5. ま と め

本論文では，CSE の構造，動作とその有効性について述べた．CSE はサーチエンジンを局所サーチエンジンとして分散し，メタサーチエンジンにより統合することにより，更新時間の短縮を実現している．たとえば，東洋大学工学部メディア・コンピューティングセンターを対象とした我々の実験では，1 時間以内に全文書のインデックスを更新できた．これは演習で作成した Web ページを同じ授業時間内に検索して確認可能なことを意味し，教育効果の向上が期待される．また，東洋大学工学部メディア・コンピューティングセンターでは，検索専用 PC サーバを用いることにより 1 分以内の更新も可能と考えられる．逆に分散サーチエンジンであることによる検索時間が長いという問題点も，先読みキャッシュの採用や検索対象サイトの絞り込みにより，無視できる範囲に収まることが確認できた．

今後は 1 ページ目の検索の高速化や，イントラネットだけでなくインターネットに対しても有効であるかどうか検証していく．

謝辞 本研究の一部は東洋大学特別研究により助成された．関係者に感謝する．

東洋大学分散システム研究室の大学院生と卒研生に

感謝する．西田喜裕君は CS を開発し，CSE の検索効率化に大きな貢献をしてくれた．山本崇君は LMSE の大部分と LSE の Gatherer を開発し，CSE の更新時間短縮に大きな貢献をしてくれた．

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) ソニー（株）：SonyDrive Search Engine. http:// www.sony.co.jp/sd/Search/
3) 萩原　淳，上原　稔，森　秀樹：分散型 Web Robot の研究，第 47 回東洋大学工業技術研究所講演会，pp.55–56 (2000).
4) 原田昌紀：サーチエンジンにおける検索結果のランキング，*bit*, Vol.32, No.8, pp.8–14 (2000).
5) Hardy, D.R., Schwartz, M.F. and Wessels, D.: Harvest User's Manual, Technical Report CU-CS-743-94, University of Colorado, Boulder (1995).
6) Kirsch, S.: Infoseek's approach to distributed search, *Report of the Distributed Indexing/ Searching Workshop* (1996). http://www.w3.org/Search/9605-Indexing-Workshop/Papers/Kirsch@Infoseek.html
7) Koster, M.: A Standard for Robot Exclusion (1994). http://info.webcrawler.com/mak/projects/robots/norobots.html
8) 國頭吾郎，相澤清晴，森川博之，青山友紀：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，マルチメディア・分散・協調とモーバイル（DICOMO'2000）シンポジウム論文集，IPSJ Symposium Series, Vol.2000, No.7, pp.157–162, 情報処理学会，ISSN 1334-0640 (2000).
9) Lawrence, S. and Gilles, C.L.: *The NECI Metasearch Engine*, NEC Research Institute. http://www.neci.nec.com/~lawrence/inquirus.html
10) Markatos, E.P.: On Caching Search Engine Query Results, *5th International Web Caching and Content Delivery Workshop* (2000).
11) 村岡洋一：「Internet 広域分散協調サーチロボットの研究開発」研究成果報告書，早稲田大学 (1999).
12) 西田喜裕，山本　崇，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キャッシュサーバの改善と評価，情報処理学会第 62 回全国大会講演論文集，pp.3–379–380 (2001).
13) 能登信晴，竹野　浩：スケーラブルな WWW 情報収集ロボットの設計と実装，情報処理学会マルチメディア通信と分散処理研究会ワークショップ論文集，pp.7–12, ISSN1344-0640 (2000).
14)（社）情報処理学会：情報処理学会. http://www. ipsj.or.jp/
15) Sato, N., Uehara, M., Sakai, Y. and Mori, H.: Distributed Information Retrieval by using Cooperative Search Engines, *21st IEEE International Conference on Distributed Information Systems Workshops*, pp.345–350, ISBN 0-7695-1080-9 (2001).
16) 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，マルチメディア，分散，協調とモーバイル（DICOMO'2001）シンポジウム論文集，Vol.2001，No.7，pp.465–470，情報処理学会，ISSN1344-0640 (2001).
17) 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，マルチメディア通信と分散処理ワークショップ論文集，IPSJ Symposium Series, Vol.99, No.18, pp.91–96, 情報処理学会，ISSN 1334-0640 (1999).
18) Sato, N., Yamamoto, T., Nishida, Y., Uehara, M. and Mori, H.: Information Retrieval Method for Frequently Updated Information System, *Databases in Networked Information Systems, International Workshop DNIS 2000*, Bhalla, S.(Ed.), LNCS, Vol.1966, pp.188–199, Springer-Verlag, ISBN 3-540-41395-2 (2000).
19) 日本電信電話（株）ソシオネットワークコンピューティング研究プロジェクト：Ingrid：広域情報検索システム．http://www.ingrid.org/index-j.htlm
20) 上原　稔，森　秀樹：最新情報の検索に適した協調サーチエンジン，*bit*, Vol.33, No.1, pp.43–49 (2001).
21) 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ー対象の最小化，マルチメディア通信と分散処理ワークショップ論文集，IPSJ Symposium Series, Vol.99, No.18, pp.85–90, 情報処理学会，ISSN 1334-0640 (1999).
22) Weider, C., Fullton, J. and Spero, S.: Architecture of the Whois++ Index Service, RFC1913 (1996).
23) 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におけるインデックス更新時間の短縮化，情報処理学会第 61 回全国大会講演論文集，pp.3–171–172 (2000).
24) 山本　崇，佐藤永欣，西田喜裕，上原　稔，酒井義文，森　秀樹：協調サーチエンジンにおけるインデクサの開発，情報処理学会第 62 回全国大会講演論文集，pp.3–231–232 (2001).
25)（株）フレッシュアイ：フレッシュアイ．http://www.fresheye.com/

(平成 13 年 6 月 5 日受付)
(平成 13 年 12 月 18 日採録)

佐藤 永欣（学生会員）

昭和 51 年生まれ．平成 13 年東洋大学大学院修士課程（情報工学）修了．平成 13 年同大学大学院博士後期課程（情報工学）入学．現在在学中．

上原 稔（正会員）

昭和 39 年生まれ．平成元年慶應義塾大学大学院修士課程（電気工学）修了．平成 5 年同大学院博士課程（計算機科学専攻）単位取得退学．同年東洋大学工学部情報工学科講師．平成 7 年慶應義塾大学大学院博士課程（計算機科学専攻）工学博士．平成 10 年東洋大学工学部情報工学科助教授．分散計算，プログラミング言語等に興味を持つ．ACM，IEEE 各会員．

酒井 義文（正会員）

昭和 42 年生まれ．平成 2 年東北大学工学部情報工学科卒業．平成 7 年同大学大学院情報科学研究科情報基礎科学専攻博士後期課程修了．博士（情報科学）．現在，東洋大学工学部情報工学科助教授．計算論的学習理論等の研究に従事．電子情報通信学会会員．

森 秀樹（正会員）

昭和 52 年慶應義塾大学大学院博士課程修了．現在，東洋大学工学部情報工学科教授．昭和 59 年から半年間訪問研究員として UCLA 計算機学科に渡米，主として，フォールトトレラントアーキテクチャ，分散処理アーキテクチャ等のコンピュータアーキテクチャに関する研究に従事．IEEE，ACM 各会員．工学博士．