協調サーチエンジンにおける分散型インデックス更新手法

宇田川 稔† 佐藤 永欣† 上原 稔† 酒井 義文‡
†東洋大学工学部情報工学科
‡東北大学大学院農学研究科

既存のサーチエンジンでは、文書収集とインデックス作成は分けて議論されてきた。そのため、文書収集の高速は広く研究されていたが、インデックス作成の高速化は、あまり議論されていない。しかし、最新情報サーチエンジンを構築するには文書収集とインデックス作成は統合して議論されるべきである。そこで文書収集とインデックス作成を統合したパイプラインに基づくインデックス更新処理方式を提案した。本文では、分散型サーチエンジンである協調サーチエンジンにおいて、分散型インデックス更新手法について説明する。

Distributed Index Updating Method in Cooperative Search Engine

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

Crawling and Indexing have been divided and discussed by the existing Search Engine. Therefore, although the high speed of Crawling was studied widely, it seldom argues about improvement in the speed of Indexing. However, building the fresh information Search Engine should unify and argue about Crawling and Indexing. Then, the Index Updating Process based on the pipeline who unified Crawling and Indexing was proposed. In the text, the technique of having distributed Index Updating Process is discussed in Cooperative Search Engine which is distributed Search Engine.

1. はじめに

組織内では公開されているWeb文書より、はるかに多い非公開のWeb文書があると考えられる。このような埋もれる情報を発掘するには組織内情報検索システムが必要である。また、このようなシステムでは、新鮮な情報を検索できることが必須の機能である。従来のサーチエンジンは1台のマシンにより文書を収集して検索を行う集中型アーキテクチャを採用している。このようなアーキテクチャでは、現在の情報の増加や多種多様の情報形態から、新鮮な情報を検索するには限界がある。そこで、我々は分散型サーチエンジン「協調サーチエンジン」(Cooperative Search Engine, CSE)[1][2]を開発した。CSEでは、個々のWebサーバにインストールされた局所的なサーチエンジンをメタサーチエンジンが統合することで、文書収集を不要とする。この結果、更新間隔の大幅な短縮に成功した[2]。しかし、すべてのWebサーバに局所的サーチエンジンを組み込むことができるわけではない。その場合、CSEの長所を十分に活かすことができない。そこで、本文では従来方式の集中型アーキテクチャで用いられているHTTPアクセスによる文書収集を効率的に行う方式について考察する。

従来から文書収集の問題は広く認知されていたため多くの研究がなされている。1つは通信遅延を考慮した分散ロボットによる分散収集[3]、ネットワークトラフィックを限界まで用いるための超並列収集[4]、学内イントラネット内での情報収集[5]などがある。しかし、これらは文書収集のみに限定した方式である。我々の経験では、サーチエンジンでは、文書収集よりインデックス作成がボトルネックとなる可能性があることがわかってきた[6]。そのため、最新情報を網羅し、適応するサーチエンジンを構築するためには、文書収集とインデックス作成の両プロセスを最適化する方法論が必要となる。この問題に対して我々は文書収集とインデックス作成の2つの処理を1つのインデックス更新処理として定義し、ボトルネックを解消しながらスループットを最大化することで対処する。その上で、協調サー

チエンジン特徴を活かし、各所に存在するWebサーバに組み込むことにより、常に最善のアクセス方法を選択する高速なインデックス更新処理を行い、全てのWebサーバにある最新情報を提供するサーチエンジンを実現することを目的とする。

本稿では、現在まで構築してきたインデックス更新のパイプライン処理を分散化する手法について説明する。構成は、2章でインデックス更新のパイプライン処理に説明し、3章では、その処理を分散化した処理について説明する。そして4章では、まとめについて述べる。

2. インデックス更新パイプライン処理

3章では、インデックス更新のパイプライン処理について説明する。まずこの処理についての概要、システムの構成、学内での実装実験、結果、考察について説明する。

2.1 概要

1章で述べたとおり、今までのインデックス更新は文書収集の高速化については議論されていたが、インデックス作成の高速化については議論されていない。我々の過去の実験[6]の文書収集はネットワークに依存し、インデックス作成はマシンスペックに依存するという結果から、スペックからプロセスを調整しながら文書収集中に収集された文書をインデックス作成することは可能であり、また、逆のことも言える。そこで、今まで文書収集とインデックス作成中で必要な処理を細分化し、それらを組み合わせて動作させるパイプライン処理を設計した。利点として、細分化したことによりボトルネックの把握や対処が容易になる。また、協調サーチエンジンの環境に対応するため、またボトルネックの解消のための分散処理の実装が容易になる。

2.2 インデックス更新パイプライン処理の構成

インデックス更新パイプライン処理の構成を図3に示す。1つ1つがアクターとなる。ほとんどが1入力1出力のプリミティブアクターとなっている。動作は、トップページを最初のアクターに入れると、データ駆動で処理を行う。次に処理対象になる文書はリンク解析により抽出され、リンクが切れるまで行われる。終了は、各アクターが休止状態であるのと各アクターをつなぐキューにオブジェクトが存在しなくなった時、終了したと見なす。

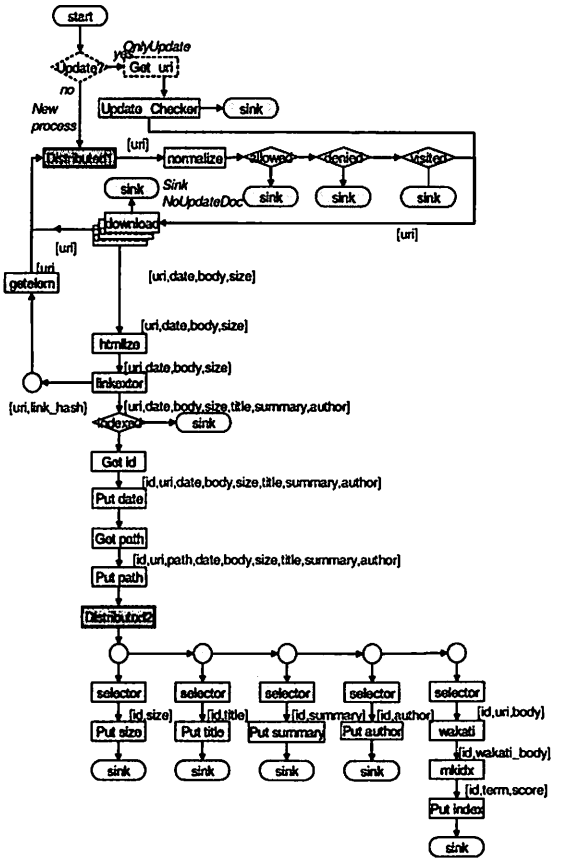

図3. インデックス更新パイプライン処理の構成

2.3 評価・ボトルネックの特定・解消

シングルプロセッサ上での1つ1つのアクターの処理時間を特定するため、まず1台のマシンを使い実験を行った。実験には、PC/AT互換機(CPU:Celeron 1.2GHz, Mem:448MB、NIC:Intel/pro 100+)のマシンを用いた。実験対象は、学内に存在するWebサーバである。まず、学内の文書を対象にインデックス更新パイプライン処理を行った結果を示す。

表1 実験結果

対象Webサーバ数	52サーバ
対象文書数	4536文書(43524KB)
平均文書サイズ	6.925KB
完全更新（並列度1）時間	1:04:58.23
スループット	1.38[doc/sec]
単純更新（並列度1）時間	4:00.83
スループット	18.85[doc/sec]
更新文書数(1週間後)	16文書

単純更新は、ほとんど更新されている文書が無かったた

め、更新されているかのチェックの時間だけで終了した。我々のシステムは最新情報を対象としているため、日に数回チェックを行う予定で、対象となるのは多くて数十文書ぐらいであることを考えれば、高速に処理できていると言える。

しかし、完全更新の結果は、過去の既存アプリケーションを用いた実験[5]の 30 分という結果から、どこかにボトルネックがあり、ロスしていると考えられる。完全更新のどのプロセスがボトルネックになっているか調べた。インデックス更新のパイプライン処理の各アクターのスループットを図4に示す。

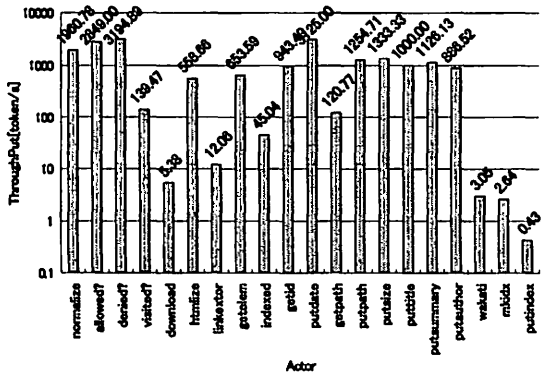

図4. 各アクターのスループット

ボトルネックになっているのは、download、linkextor、wakati、mkidx、putindex である。それぞれ、文書収集、リンク抽出、分かち書き、インデックス作成に対応するアクターである。

本稿では、文書収集におけるボトルネックの解消を説明する。文書収集は、全体から見るとボトルネックになっているが、文書収集は過去の実験[5]でマシンスペックにはあまり依存せず、ネットワークの状態に依存するといったことが分かっている。そこで、文書収集に対しては、多重化により対処するようにした。文書収集を多重化した結果のスループットの向上を図5に示す。

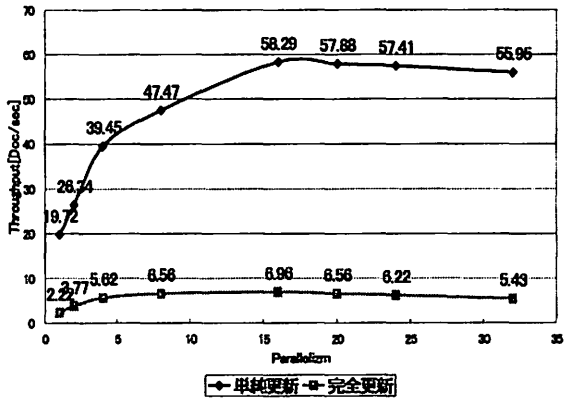

図6. 文書収集の多重化スループット

多重化によりスループットの向上が見られた。また、並列度1で単純更新では4分という結果が得られたが多重化により1分20秒(スループット58.29[doc/sec])まで減らすことができた。しかし図5から16並列あたりで頭打ちになっているのが分かる。そこで、文書の多いWebサーバを対象にどのアクターがどの時間にコネクションをつないでいるかを調べた。結果から常に同じような周期で全てのアクターが同じ Web サーバにアクセスしようとしているのが分かった。これは、web 文書の特色だと言える。実験後のリンク解析からも言えたが、html で web 文書を作成し、リンクを張るとき他の Web サーバに張るより、自身の Web サーバにある文書にリンクを張ることが多い。これによりその時間に1つの Web サーバに集中し、16 並列あたりで頭打ちになってしまう。そこで我々は、図5より1つの Web サーバには 16 並列まではスループットの向上が得られることを考慮する。しかし、それぞれの Web サーバは文書数の違いから、ただ単に16並列すればいいものではない。そこで、新しい Web サーバが出現するごとに、その Web サーバ専用の download アクターを起動する。そして、ここのキューに設定した値より URI がたまった場合、新たにその Web サーバ専用の download アクターを起動する。このようにしてその Web サーバの文書数に合わせて並列度を増やしていく手法を行う。

3. 分散型インデックス更新過程の設計

図4より、各アクターのスループットは、均等化されていない。これによりボトルネックが発生し、ロスを生じることが予想できる。我々はこの問題に対し、前章でいくつか紹介した多重化やモジュールを使用することにより、ボトルネックとなるアクターを特定しスループットを向上することで解消してきたが、それだけではボトルネックの解

消にならず、分散処理についての手法を考えた。

3.1 パイプライン処理における分散ポイントの特定

我々が設計したインデックス更新のパイプライン処理の中で分散を行うポイントとして、2ヶ所考えられる。(図3参照)最初のポイント(以下 Distributed1)は、normalize の前で1番にオブジェクト(URI)が送られるポイントである。ここのポイントではリンク解析され、抽出された URI が最初に流れるポイントである。このポイントで新しい Web サーバの URI が検出された場合、他のマシンに URI を渡すようにする。このことにより、各所に散らばったマシンからその URI の Web サーバに最も通信遅延が小さいマシンに処理を割り振ることも可能で最適化された文書収集が可能になる。次のポイント(以下 Distributed2)は、各要素(サイズ、タイトルなど)をデータベースに格納する前である。なぜこのポイントが必要かというと1番の要因はインデックス作成の工程(分かち書き、インデックス作成、スコアデータベース格納)のボトルネックである。この工程は、大きなボトルネックでありマシンパワーが大いに依存する。このことによりマシンパワーが比較的空いているマシンに割り振ることが必要になる。文書のインデックスが分散していることは、検索時に影響があると考えられるが、協調サーチエンジンの特性上[1]、どの場所にインデックスが置かれても影響がない。他のサイズやタイトルなどのデータベースの格納などの前にポイントを置いたのは、文書 ID の整合性を考慮したからである。Distributed2 のポイントで分散する場合、文書に対するコスト計算が必要になる。このコストを考慮することで分散処理を最適化する。コストを次のように定義する。インデックス更新のパイプライン処理は、全ての処理が文書に依存する。しかし、文書といっても、文書 ID、URI、サイズ、通信遅延など各アクターにより依存するものが違う。よって、それぞれの要素を含む文書配列 X を以下のように定義する。j は文書 id とする。

$$X_j = (id, uri, path, date, body, size, title, summary, author \cdots) \quad j = 1,2,\cdots m$$

Distributed2 では、インデックス作成の工程の各文書に対するスループットにだけ考慮する。Distributed2 を通る文書 Xj の配列は、文書サイズを保持する。この文書サイズとその文書の実験より得られたアクターの処理時間用いれば、この文書がインデックスを作成するのにどれだけコストがかかるか予測することができる。実験結果より、平均データサイズは 6.925[kB],インデック作成工程の平均処理時間は 2.6[s]であるので、

$$Cost(X_j) = \frac{Size_j * 2.6}{6925} [s/B]$$

として、コストを計算する。このコストを用い、各マシン内でボトルネックにならない上限値 $Cost_{upper}$ を設定し、分散処理を図る。

3.2 Distirbuted1&2 ポイントにおける分散処理

実際に各ノードマシンを協調することを考える。どのノードが処理を行うのに空いているのか、どの Web サーバを処理しているのかという情報を各マシンが共有するために、TupleSpace を用いる。TupleSpace[6]とは、分散システムのための分散共有メモリである。Distributed1 と Distributed2 とも TupleSpace に Tuple を投入する Write アクター、取り出す Take アクター (読み込む read を含む) アクター、これら2つを制御する Locate アクターからなる。Locate アクターは他の download アクターや mkidx アクターなどと同様にデータ駆動で動作する。Write アクターと Take アクターは、Locate アクターからの実行要求に対し動作をする。Locate_DB とは、どのノードがどの Web サーバを処理しているかを登録しているデータベースである。

TupleSpace を用いた分散型インデックス更新過動作を Distributed1 と Distributed2 の TupleSpace への Tuple の投入、取り出しの動作について説明する。

Distributed1 における Tuple 投入動作(図7参照)

1. Locate アクターは、キューから先頭の URI を取り出す。
2. その URI の Web サーバが処理されている Web サーバを判定するために Locate アクターは Locate_DB にアクセスする。

3-1-1. どこのノードでも処理されていない場合、この URI を Write アクターに送る。

3-1-2. Write アクターは、[送信元ノード名、送信先ノード名、URI、新しい Web サーバを示す'new']の配列を Tuple とし、[hostname,nil,URI,'new']を投入する。この場合、送信先のノード名は指定しない。

3-2-1. 他のノードで処理されている場合、この URI を Write アクターに送る。

3-2-2. Write アクターは、[送信元ノード名、送信先ノード名、URI、nil]の配列を Tuple とし、[hostname,to_Node,URI,nil]を投入する。

3-3. この URI の Web サーバが同じノードで処理されている場合、同じノードの次のアクターに送る。

4. 1に戻る。

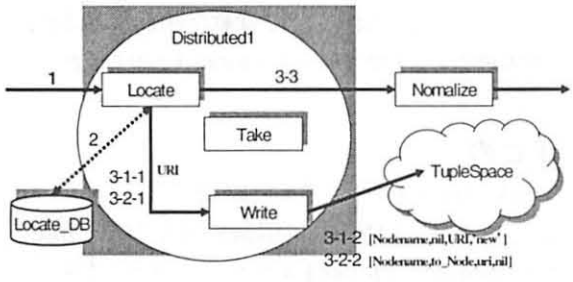

図7. Distirbuted 1 における Tuple 投入

Distributed1 における Tuple 取り出し動作(図8参照)

1. Take アクターは待機状態にあり、Locate アクターから一定の間隔で実行要求が送られ動作する。

2. Take アクターは 3 つの Tuple を待つ。そのうち 2 つが「新しい Web サーバの Tuple」と「自分のノード宛ての Tuple」であり、この 2 つが無いか、TupleSpace に問い合わせをする。

3-1. この 2 つにマッチする Tuple があった場合、Tuple を取り出し、URI を次のアクターに送る。

3-2. これを Locate_DB に登録する。

3-3. 「自分がこの Web サーバを処理しています」ということを知らせるために、Write アクターに Web サーバ名を送る。

3-4. Write アクターは、[送信元ノード名、送信先ノード名、URI、全ノードに知らせることを示す'AllNode']の配列を Tuple とし、[Nodename,nil,host,'AllNode']投入する。

4. 1 に戻る。

(他のノード)

□. すべてのノードは、Take アクターは前の 2 つの Tuple の他に 3 つ目の Tuple である「どこのノードで処理されているか」を表す Tuple を待つ。マッチする Tuple があった場合、Tuple を読み出す。

□. この Web サーバ名と処理されているノード名の配列である[HostName,Node_Name]を Locate_DB に登録する。

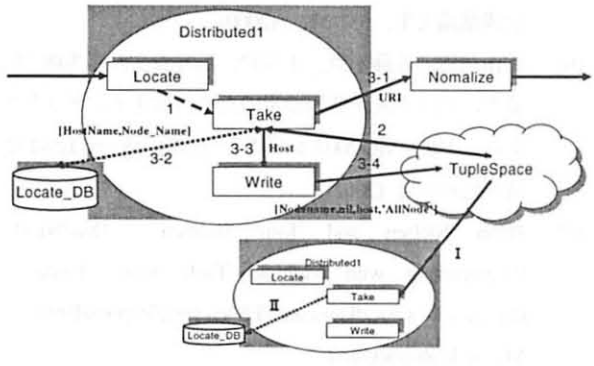

図8. Distirbuted 1 における Tuple 取り出し

Distributed2 における Tuple 投入動作(図9参照)

1. Locate アクターは、キューから先頭の文書 Xj を取り出す。

2. Locate アクターは、文書 Xj のサイズからコストを計算する。それを、All_Cost に足す。

3-1-1. 上限値を超えていなければ次のアクターに渡す。

3-1-2. その文書のインデックスを作り終えた後にその文書のコストを引く。

3-2-1. 上限値を超えている場合は、文書 Xj を Write アクターに送る。

3-2-2. [送信元ノード名,nil,文書 Xj,インデックス作成を表す'index']の配列を Tuple とし、[NodeName,nil,Xj,'index'] を TupleSpace に投入する。

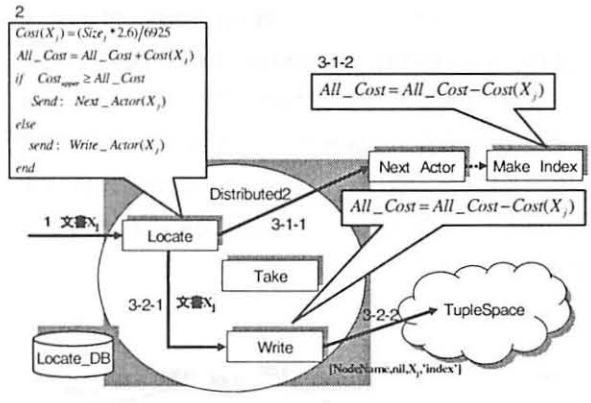

図9. Distirbuted2 における Tuple 投入

Distributed2 における Tuple 取り出し動作(図 10 参照)

1. Take アクターは待機状態にあり、Locate アクターから一定の間隔で実行要求が送られ動作する。つまり Distributed2 での Locate アクターは、コスト計算により上限値より小さいとき、Take アクターを動かす。

2. Take アクターは、TupleSpace にアクセスする。

3. マッチする Tuple があった場合、その Tuple を TupleSpace から取り出す。

4-1. その Tuple(文書)のコストを足し、それを同じノードの次のアクターに渡す。

4-2. その文書のインデックスを作り終えた後にその文書のコストを引く。

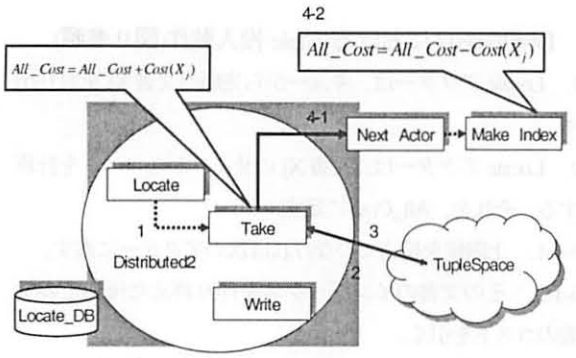

図 10. Distirbuted2 における Tuple 取り出し

3.3 評価·考察

このシステムを用いた場合、マシン数を増やすことでどれだけの処理時間短縮、スループットの向上が得られるか実験を行った。実験には, 9 台の PC/AT 互換機(CPU:Celeron 1.2GHz, Mem:448MB、NIC:Intel/pro 100+)で構成された並列計算機の環境を用いた。実験対象は、シングルプロセス同様に学内に存在する Web サーバである。各マシンの文書収集の並列度はもっともスループットの高かった 16 とする。以下に、クラスタマシンを用いた分散型インデックス更新実験の結果を示す。

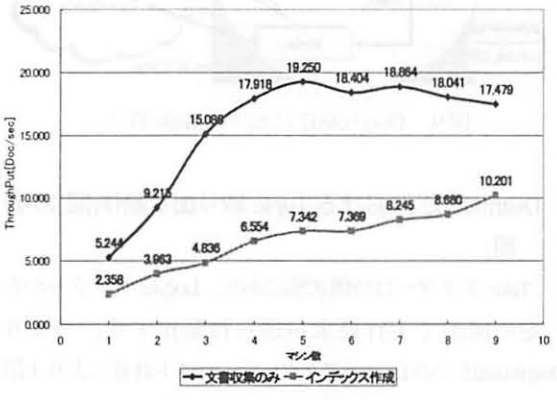

図 11. 分散型インデックス更新実験結果

図 11 より、ノード数を増やすことにより文書収集はスループットの向上が見られるが、一定のところで頭打ちになっているのが分かる。この理由として考えられるのが、それぞれ割り振られる Web サーバの文書数の違いが考えられる。各ノードは動的に処理を行っているためどの Web サーバの文書数が多いか分からない。これにより、文書数の少ない Web サーバを処理している幾つかのノードには待ち時間が生じてしまう。しかし、インデックス作成までの処理をした結果を見ると、その待ち時間を他のマシンが TupleSpace により共有されたインデックスの作成することで解消している。これにより、柔軟な処理が実現され、ノードを増やすことでスループットの向上が見られた。

4. まとめ

今回は、分散インデックス更新過程をクラスタマシンに実装し、学内ネットワークでの計測を行い、処理を分散することでスループットの向上することができた。今後、ノードを増やすことでどれだけスループットを向上できるかという予測のためにこのシステムを定式化を行う。そして、システムのスケーラビリティを考慮して、TupleSpace を多重化し他のドメインにあるマシンとも通信ができるようにシステムを設計し、実装を行う。

参考文献

[1] Nobuyosi Sato, Minoru Uehara, Yosibumi Sakai, Hideki Mori, "Fresh Information Retrieval in Cooperative Search Engine, " In Proceedings of the ACIS 2nd International Conference on Software Engineering , Artificial Intelligence , Networking & Parallel/Distributed Computing(SNPD'01), pp.104-111, (2001.8.20)

[2] Takashi Yamamoto, Nobuyosi Sato, Yoshihito Nishida, Minoru Uehara, Hideki Mori "Updating Information in Cooperative Search Engine," DICOMO2000 IPSJ Symposium Series,Vol.2000, No.7, pp.307-312 (2000.06)

[3] 山名早人等 「分散型 WWW ロボットによる WWW 情報収集」 DEWS'98 、http://www.etl.go.jp/~yamana/Publications/ABST/DEWS98/24.htm

[4] 能登信晴等 「スケーラブルな WWW 情報収集ロボットの設計と実装」 DPSWS (2000)

[5] 大島利充、高井昌彰「マルチスレッドを用いた学内限定 Web 検索ロボット」情報処理学会第 62 回全国大会講演論文集、 3T-07、 (2000)

[6] 宇田川稔, 佐藤永欣, 上原稔, 酒井義文, 森秀樹 "組織内における最新情報検索のための高速インデックス更新", DICOMO'2002 シンポジウム論文集, pp.129-132, 情報処理学会 (2002)

[7] Brian Nielsen and Tom Slrensen 「 Distributed Programming with Multiple Tuple Space Linda 」 http://www.cs.auc.dk/research/FS/teaching/Reports/1993/Mts-linda.abstract.html