インデックス更新のためのパイプライン化手法

宇田川 稔、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
{ti980039,jju}@ds.cs.toyo.ac.jp, {uehara,sakai,mori}@cs.toyo.ac.jp

既存のサーチエンジンでは、インデックス作成処理は文書収集と分離して実行されていた。インデックス作成処理は、文書収集より重いため、ボトルネックとなる。したがって、インデックス作成と文書収集は統合されるべきである。そこで、我々はパイプラインに基づく処理方式を提案する。本方式では一連の処理をマクロデータフローに基づく並行プロセスの集合でモデル化する。この方式はボトルネックを容易に推定できるという特徴を持つ。本文では、本方式を用いてインデックス更新の全過程におけるボトルネックの特定について述べる。

A Pipeline Processing Method for Index Update

Minoru Udagawa, Nobuyoshi Sato, Minoru Uehara, Yoshifumi Sakai, Hideki Mori
Dept..of Information and Computer Sciences, Toyo Univ., Japan

In traditional search engines, indexing process is separated from crawling. Indexing often becomes bottleneck because indexing a file is heavier than fetching a file. Therefore both indexing and crawling processes should be integrated. So, we propose a pipeline based processing method, in which sequence of processing is modeled as a set of macro dataflow actors. It is easy for this method to detect the bottleneck. In this paper, we describe about a bottleneck detection method using pipeline model.

1. はじめに

インターネットの普及に伴い、Web ページの検索が重要になってきた。Web ページの検索はサーチエンジンによって行われる。既存のサーチエンジンは集中型アーキテクチャに基づく。集中型アーキテクチャでは、ロボットが文書を収集し、インデクサがそれらの文書からインデックス（転置ファイル）を生成する。このような方法では、更新時間が長くなり、新鮮な情報を検索することが困難である。ここで、更新時間とは、文書が公開されてからその文書を検索できるようになるまでの期間である。分散型アーキテクチャに基づくサーチエンジンを分散型サーチエンジンという。分散型サーチエンジンでは、各 Web サーバ上で局所的にインデックスを作成するため、更新時間が短い。そこで、我々は新鮮な情報を検索するために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)」[1]を開発した。

協調サーチエンジンでのインデックス更新処理では、組織内に対応した文書収集アクセス

法により高速収集を実現してきた。しかし、インターネットに対応した方法については、マシン負荷が大きかったり、タイムロスが生じたりと必ずしも最適化されているとは言えなかった。そこで本研究では、今までの処理方法を見直し、今まで別プロセスとされていた文書収集とインデックス作成を併せたインデックス更新処理を1つ1つの最小の単位として捉え、パイプライン化を図る手法について提案した。

各プロセスをスレッドにより細分化することにより、ボトルネックの把握ができ、ボトルネックの解消も容易になると考えられる。また、各プロセス単位で並列処理や負荷分散も可能になる。

本文の構成は以下の通りである。2章ではCSE の原理について述べる。特に、更新ならびに検索時の動作について述べる。3章では、関連研究について述べる。4章では、研究にいたる準備実験について述べる。5章ではストリーム計算モデルについて述べる。パイプラインを最適化する理論について述べる。6章ではインデックスの更新のパイプライン処理について述べる。7章では、評価について述べる。最後にまとめを述べる。

2. 協調サーチエンジン

CSEは以下の部品から構成される(図1参照)。

- 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(Query based Site Selection、 QbSS)を行う。
- 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。サイト選択の結果を保存するキャッシュをサイトキャッシュ(SC)と呼び、検索結果を保存するキャッシュを検索キャッシュ(RC)と呼ぶ。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 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 1 の Indexer)、検索 (図 1 の Engine) を行う。

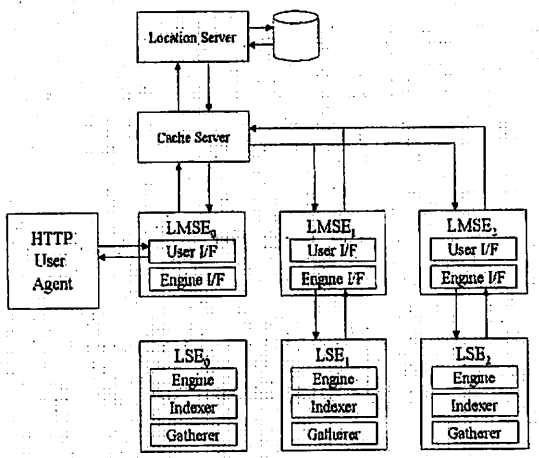

図 1. CSE の概要

更新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. LMSE は必要ならば 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 にメタインデックス(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。

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 LMSE0 に検索を依頼する。
2. LMSE0 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 LMSEi を選び、検索要求を並行に送信する。
6. LMSEi 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 LMSE0 へ返す。
9. LMSE0 は結果を HTML に整形し、ユーザへ返す。

3. 関連研究

文書の高速収集に関する研究はいくつか行われている。

早稲田大学の山名は、インターネットに対応した分散型 Web ロボットを開発している。これは、対象 Web サーバに対し、回線距離などを考慮し、各 Web ロボットに割り振り、協調して高速収集を実現するものである。[2]

NTT サイバースペース研究所の能登は、複数のマシン、プロセス、スレッドによる多重化により、利用できる回線帯域を上限として収集速度を必要に応じて向上できるスケーラビリティを持った Web ロボットの設計を提案している。[3]

北海道大学の大島は、学内イントラネットに対応し、ローカルのサーバ群の情報は、情報の鮮度が重要になる点を考慮し、高速な文書収集ロボットの開発を行っている。学内の文書収集のスループットで 26.5[file/s]という研究結果を発表している。[4]

しかし、これらの研究では文書収集のみのスループットである。インデックス更新には、文書収集に加え、インデックスの生成が必要である。次に実際のサーチエンジンのインデックス更新を紹介する。

Google[5]は、今日存在するサーチエンジンの中で、世界最大の情報を持ち、ヒット率も高いサーチエンジンとして知られている。Google は約 8000 台を用い、クラスタコンピューティングにより 7000 万件/日の検索を 1 件あたりほぼ 0.5 秒で処理する。しかし、Google は、問題点も存在する。それは、更新時間の遅さである。Google では、ある Web ページが更新された場合、実際の検索に反映されるのに 60~90 日を要していると言われている。

CSE では組織内の Web サーバに高速にアクセスする方法として、現在まで以下の方法を取り入れ考えてきた。直接アクセス方法は、NFS を含むファイルシステムによりアクセスする方法である。文書収集を考えることなくインデックス作成処理が行え、最も高速な方法であると言える。アーカイブアクセス方法は、リモートの Web サーバに CGI を用いて複数のファイルをアーカイブかつ圧縮して転送したものを受信側で解凍する。アーカイブ、解凍に時間を要すると考えられるが、複数のファイルを転送するのでファイル当たりの通信遅延は比較的小さくなると言える。最後に、一般的なサーチエンジンのアクセス方法として用いられる HTTP アクセスがある。前 2 つは、実現すると高速な文書収集を実現できる。しかし、対象の Web サーバにアカウントのない場合、やはり HTTP アクセスで収集を行う必要がある。

4. 準備実験

本研究の準備実験として、並列収集可能な収集ツール Httrack を用いて本学に存在する TOP サーバ内の文書に対し収集実験した。過去の収集結果からこの Web サーバには約 800 文書存在している。実験マシンは、マシン A（CPU : Cerelon 300MHz、Memory : 128Mbyte、NIC:Intel/pro 100+)と、マシン B(CPU:Cerelon 1.2Hz、Memory:448Mbyte、NIC:Intel/pro 100+)を用いた。図 2 にマシン A

およびマシン B を使った並列収集の結果を示す。

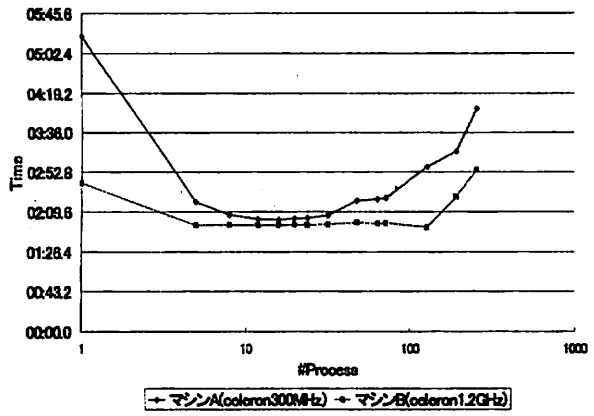

図 2. マシン別並列処理結果

図 2 より、マシン A とマシン B はあまり収集時間に変化はない。また、マシンスペックから最速の並列数は違うもののどちらのマシンでもただ増やしただけでは、収集効率が落ちてしまう。これは、収集処理が、マシンよりネットワークに依存していると言える。

インデックス作成は、マシンスペックがボトルネックになっていると言える。これは、マシンスペックをあげれば、処理効率も上がる見込みがあるということである。次の実験では、マシン B を使い、文書収集とインデックス作成を並行して行う実験をした。インデックス作成には、Namazu を用いた。単独で計測した結果と共に図 3 に示す。

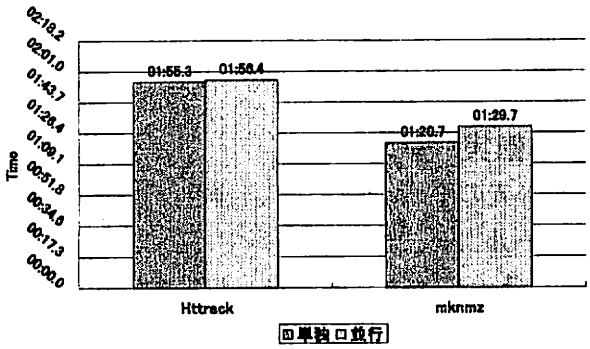

図 3. 並行処理結果

この結果から、文書収集とインデックス作成は、それぞれネットワーク、マシンスペックに依存し、互いに影響しあわないことが分かる。

ここで処理単位について考える。収集やインデックス作成を行うとき、文書単位か Web サーバ単位に分けられる。

文書単位であると、1 つ 1 つのコストも小さく、並列処理や分散処理の設計がしやすくなる。しかし、収集を考えるとどの Web サーバにもリンクを辿るので、遠い Web サーバも処理対象になり無駄な通信遅延が発生することが考えられる。Web サーバ単位の利点は、収集を行うとき、リンクは同 Web サーバ内の文書しか対象としないため、ファイル当たりの通信遅延も小さくなる。

そこで、本学 4 キャンパス内の 54 Web サーバ、5570 ファイルを対象とし、多くのサーチエンジンで用いられている Wget や Httrack に代表されるロボット、Namazu、Freya、SGSE などのインデクサのアプリケーションを用いて、クラスタマシンを使った実験を行った。クラスタマシンの各ノードへの割り振りには、割り振り時に余メモリ容量が空いているノードに Web サーバ単位で過去に収集された文書容量と通信遅延から割り振った。実験の環境は、5 台のマシンをクラスタ OS-SCore を用いて、サーバマシン 1 台（CPU:Pentium Ⅲ 886MHz、Mem:128MB、NIC:Intel/pro 100+）、ノードマシン 4 台（CPU:Celeron300MHz、Mem:128MB、NIC:Intel/pro 100+）を用いた。この実験結果から、Web サーバ単位であると、処理コストをどう組み合わせても、ノードごとの均一なコストにするのは困難であり、完全な負荷分散の実現は難しくなる。よって、本研究では、Web サーバ単位であると時間ロスが生じてしまうと考えられるため、今後は文書単位以下での処理を基本として考えていく。

5. ストリーム計算モデル

ストリーム計算モデルはマクロデータフローの一種である。青柳氏のストリーム計算モデル[6]の特徴は、通信路であるストリームにおいて複数のトークン（オブジェクト）が混在することを許し、かつ到着順序を保証することで

ある。構成要素は、計算の実行主体であるアクターと通信路であるストリームの組み合わせで構成される。アクターは、資源を抽象化したオブジェクトである。アクターの動作は

1. 実行再開待ち
2. 入力ポート集合からの受信
3. アクターの実行本体
4. 出力ポート集合への本体

の一連の作業を行っている。図4(a)はアクターの動作を表す。実行時間 Tr、待ち時間 Tw と表すと、このような周期で実行されるプロセスのスループット（単位時間当たりの平均出力）は、1/(Tr+Tw)[token/s]となる。したがって、スループットを向上させるには Tw を 0 に近づけることが重要となる。

また、複数のアクターにおけるパイプラインを図4(b)に表す。個々の矢印は各プロセスの処理時間を示し、その輪が全体のプロセスを示す。例では、3 ノードにそれぞれ 1 つずつプロセスが配置されている。このような全体プロセスのスループットは、$\min_{0<i\leq n} 1/(Tri + Twi)$ [token/s] となる。このとき最小のスループットを持つプロセス p がボトルネックとなる。スループットを向上させるには個々の Twi を 0 に近づけることと Tri を減らすことが重要である。

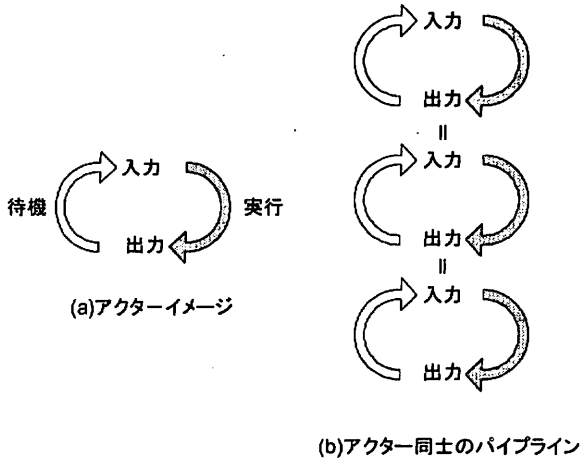

図 4. プロセスとパイプラインの種類

6. インデックス更新パイプライン処理

ストリーム計算モデルを基礎とし、設計したインデックス更新をパイプライン化した構成を図4に示す。1つ1つがアクターになり、ほとんどが 1 入力 1 出力のプライミリティアクターとなっている。ストリームは、1マシン内で行われるため考慮せず、各アクターは、互いにキューで結合され、データ駆動で動作する。プログラムは、オブジェクト指向言語である Ruby を用いて構築し、各プロセスをスレッドにより細分化することによって、ボトルネックの把握が容易になる。

- 楕円は、タスクの始終端を表す。
- 四角は、実行プロセスを表す。
- ひし形は、分岐を表す。例えば、文書型の一致、対象ではないホスト、収集された文書の確認をとる。
- ○は、スレッドによる並列処理を表す。独立したアクターは逐次にする必要がなく、並列処理を行っている。
- 矢印は、処理の流れを表す。

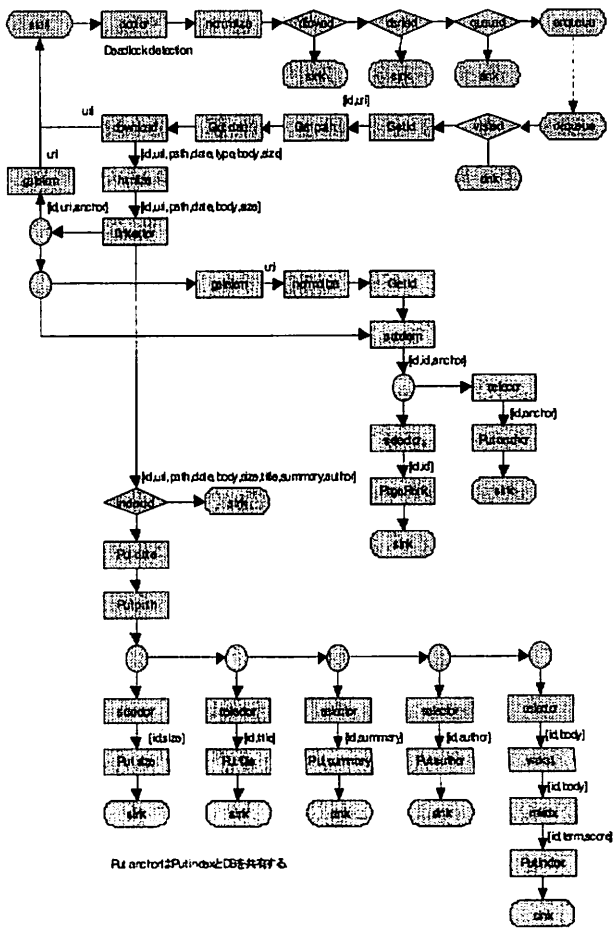

図 5.インデックス更新のパイプライン

7. 評価

今回設計したインデックス更新のパイプラインモデルのボトルネックを特定するため、計測を行った。対象は本学に存在するWebサーバである。計測には、マシンC(CPU:Dual-Pentium Ⅲ933MHz、Mem:1GB、NIC:Intel/pro 100+)を用いた。

インデックス更新の各アクターの1回の呼び出し当たりの平均処理時間を図6に示す。

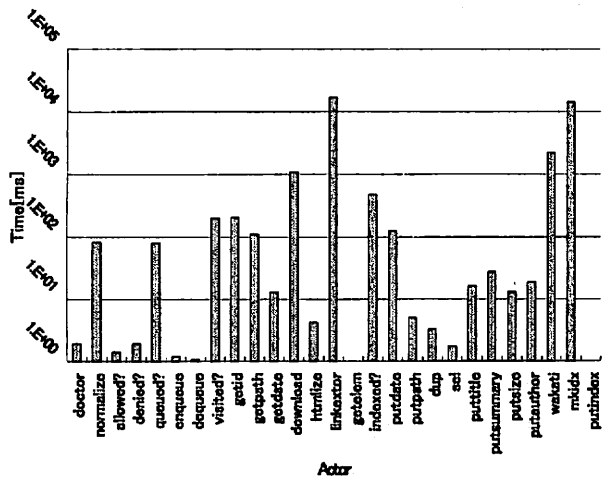

図6.インデックス更新過程の処理時間

図6より、ボトルネックになっているのは、download、linkextor、wakati、mkidx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。

まず、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、1文書当たり約1秒であり、これ以上の短縮は難しい。リンク抽出とインデックス作成は、行単位で繰り返してタグ、語の解析処理を行っているため、処理時間が増えていると考えられる。分かち書きは、1文書ずつ分かち書きの茶筅アプリケーションを呼び出し、文書の本文を一時、ファイル出力しているため、他のプロセス起動の時間がロスを生んでいる。そこで、分かち書きをRuby用茶筅モジュールに変えた。分かち書きのボトルネック解消結果を表1に表す。

表1.　分かち書きのボトルネック解消

	茶筅モジュール	茶筅プロセス
処理時間[ms]	9.73	2198.19

茶筅モジュールを使うことで、大幅な短縮が出来た。これは、一時ファイル作成、またプロセスとして茶筅を動かないので、起動にかかるロスを解消したことになる。

8. まとめ

今回は、インデックス更新のパイプライン化を図ることにより、処理上でのボトルネックを特定することができた。今後、このボトルネックの解消を中心に、前研究結果である各Webサーバに対するネットワーク遅延の考慮、またクラスタマシンでの並列処理も視野に入れて、プロセス間の通信コストについて考えていきたい。

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] 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

[3] 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)

[4] 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)

[5] 山名早人、近藤秀和「サーチエンジンGoogle」IPSJ Magazine Vol.42 NO.8 Aug. 2001

[6] 青柳 洋一、上原 稔、森 秀樹、佐藤 章「GAを用いた予測的タスク割付方式」電子情報通信学会論文誌 D-I Vol.J81-D-1 No.11 pp.1181-1189 1998年11月