スケーラブルな WWW 情報収集ロボットの設計と実装

能登 信晴 † 竹野 浩 ‡

† NTT サイバースペース研究所
noto@isl.ntt.co.jp

‡ NTT サイバーソリューション研究所
takeno@aether.hil.ntt.co.jp

本論文では、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もった WWW 情報収集ロボットの設計を提案する。収集速度向上の鍵は WWW サーバからページデータを受信する処理の多重度を向上することであり、スケーラビリティ実現の鍵は多重化された処理間の協調コストを下げることである。我々は、提案する設計に基づきプロトタイプを実装した。このプロトタイプのパラメータを変化させながら収集実験を行い、性能特性を把握するとともに、設計の有効性を確認した。

Design and Implementation of a Scalable WWW Information Collection Robot

TOKIHARU NOTO † AND HIROSHI TAKENO ‡

† NTT Cyberspace Laboratories
noto@isl.ntt.co.jp

‡ NTT Cyber Solution Laboratories
takeno@aether.hil.ntt.co.jp

In this paper, we propose a design of WWW robot which enables it to get the scalability to improve the collection speed on demand upto the limitation of an access line bandwidth. The key to improve the speed is how to make more multiplexity of WWW page reception from WWW servers, and the one to achieve the scalability is how to reduce the cooperation cost among the mulptiplexed reception. We implemented a prototype based on the design and carried out WWW information collection experiments with several configurations. We investigated its performance character through the experiments and made sure of the efficiency of the proposed design.

1 WWW ロボットの動作と性能要件

WWW 検索エンジンには、「ロボット」や「クローラ」と呼ばれる WWW ページを収集するプログラムが必要とされる。本論文では、これを「WWW ロボット」と呼ぶ。一般に WWW 検索エンジンでは、インデクサと呼ばれるプログラムが収集された WWW ページを入力として全文検索を行うためのインデックスを構築し、これが検索サービスに利用される。

WWW ロボットの基本動作は、以下の通りである。

1. 収集の起点となる URL を得る
2. 未収集の URL であるか確認
3. 収集が許可された URL であるか確認
4. URL が指し示す WWW ページを WWW サーバから取得
5. 取得した WWW ページから URL を抽出
6. 抽出された URL に関して 2〜5 を繰り返す

収集された WWW ページが検索可能になるまでには、インデキシングなどの処理に一定の時間がかかる。多くの WWW ページは頻繁に内容が更新されている。したがって、収集開始から検索実行までの時間が長くなると、実際の WWW ページの内容と検索結果が示す内容との間に差異が生じてしまう可能性が高くなる。

WWW ページを収集する立場から見ると、この差異をできるだけ小さくして新鮮な検索結果を提供し、また検索対象とできる WWW ページも

増やしたいと考える。そのため、短時間に大量の情報を収集できることが必要とされる。つまり **収集速度** が重視される。

しかし、WWW サーバを運営する側の立場から見ると、同一のサーバから間隔をあけずに、あるいは同時に複数の接続を行って WWW ロボットが WWW ページを収集すると、サーバやネットワークに大きな負荷がかかり迷惑である。これを避けるためには、1 つのサーバには同時に 1 だけ接続し、接続と接続の間にはなるべく長い時間を設定できる方が良い。ここでは、この時間を **訪問間隔** と呼ぶ。

したがって、訪問間隔を短くしないで、収集速度を高めることが WWW ロボットに求められるので、この 2 つを性能指標とする。ただし、本論文では訪問間隔を一定の値に定めたとき、いかに収集速度を向上させるか、ということについて検討するので、収集速度を WWW ロボットの主たる性能指標として取り上げる。

他に、WWW ロボットに求められる性能としては、未発見の WWW ページを発見する効率、過去の更新履歴から次の更新時期を推測して必要なタイミングに必要な WWW ページを収集することで無駄な収集を防ぐ能力なども挙げられるが、本論文ではこれらの性能については議論しない。

2 性能向上およびスケーラビリティ実現の方針

インターネットで公開されている World Wide Web (WWW) のページ数は年々増加の一途をたどっており、多くの検索サービス提供者がこの増加に対応すべく競争を続けている。したがって、WWW ロボットにはこの増加に対応して性能を向上できる能力が必要とされる。

WWW サーバから 1 つの WWW ページを収集する際の処理時間のほとんどは、

- DNS 検索: WWW サーバのホストネームから IP アドレスを得る
- ページデータ受信: WWW サーバとの通信コネクションを確立してデータを受信する

の 2 つによって占められる。

DNS 検索とページデータ受信にかかる時間は、WWW ロボット側をいくら改善しても、WWW サーバや DNS サーバ側の処理能力、サーバとロボットの間の通信路の条件といった外的制約があるため、ある程度以上短くできない。したがって、収集性能を向上する鍵はこれらの処理の多重度を高めることにある。ただし、1 つの WWW サーバに対して同時に複数の接続を行って収集すると相手のサーバに過度な負荷を与えてしまうので、1 つのサーバには複数の接続を同時に行わないことが WWW ロボットの一般的なルールとなっている。

また、WWW ロボットとインターネットの間にある回線容量が性能のハードリミットになる。1 WWW ページあたりの平均サイズは 7.5 K バイトであり*、この統計値に基づくと回線容量が 1Mbps の場合、

$$\frac{1000(\mathrm{Kbps})}{7.5(\mathrm{Kbyte}) \times 8(\mathrm{bit})} \times 60(\mathrm{sec}) \times 60(\mathrm{min}) \times 24(\mathrm{hour}) = 1440000(\mathrm{pages/day})$$

となり、一日で収集できるページの上限は約 150 万ページとなる。

以上をまとめると、WWW ロボットのスケーラビリティは、回線容量の制約を上限として、DNS 検索とページデータ受信の多重度を必要に応じて向上できる能力といえる。

3 関連研究

Brian Pinkerton は文献 [1] で、WWW ロボットとそれを利用した WWW 検索エンジンのの基本的な枠組みを提案している。また、Sergey Brin らは文献 [2] で大規模な WWW 検索エンジンの構築について論じている。しかし、WWW ロボットのスケーラビリティについては触れられていない。また、文献 [2] でも指摘されているように、商用ポータルサイトで利用されるような大規模検索エンジン技術に関する論文は少ない。

文献 [3], [4], [5] では WWW ロボットを広域分散配置して収集性能を向上することを提案している。しかし、我々の実験では、インターネット上の 1 点から収集する際でも、インターネットとの接続に 100Mbps のような帯域が用意されている場合、その帯域が収集性能向上の制約となるよう

*jp ドメイン以外の WWW サーバから提供されるページについては日本語で記述されたページだけを収集し、jp ドメインの WWW サーバから提供されるページについては無条件で収集するという条件下で 1000 万ページを収集した際の値である。

な WWW ロボットを構成することが困難であることがわかっている。

4 スケーラブル・アーキテクチャの設計

4.1 多重度を上げる方法とその制約

以下本稿では、DNS 検索とデータ受信の組を「収集基本処理単位」と呼ぶことにする。収集基本処理単位の多重度を高めるには下記の選択肢があるが、それぞれ制限がある。

- 1 プロセス内でマルチスレッドを利用
- 1 マシン内でマルチプロセスを利用
- 複数マシンの利用

1 プロセス内でマルチスレッドを利用すると、スレッド間のデータ共有がしやすいというメリットがあるが、多くの OS では 1 プロセス内で利用できるファイル記述子の数に制約がある。この制約を越えるためには、マルチプロセスを利用する必要がある。1 マシンで利用できるプロセスの数はマシンに搭載されたメモリ容量に制約を受ける。

一般に 1 マシン上で CPU やハードディスクなどの資源が複数プロセスやスレッドで共有されるとき、プロセスやスレッドを増やしていっても共有資源に関する競合が原因で、処理の効率が上がらないということが起こり得る。

さらに 1 マシンで実現可能な多重度を越えるためには、複数のマシンを利用する必要がある。マシン数が増えると、マシン間の通信量と通信にかかる処理が制約となって、性能が向上しないこともある。

したがって、収集基本処理単位の多重度を制約無く向上させるためには、マルチスレッド・マルチプロセス・マルチマシンの全ての形態に対応し、かつ性能向上を妨げるような問題を回避する必要がある。

4.2 URL の配分方法

WWW ロボットには URL のリストが起動時に与えられ、これを収集するとともに、収集されたページの中にハイパーリンクとして埋め込まれた URL を抽出して、その URL についても収集を行う。このように、起動時に与えられた URL と、収集されたページから発見された URL を、どの収集基本処理単位に割り当てるかということが、処理単位間の協調コストになる。多重度と性能の間にスケーラビリティをもたせるためには、このコストを下げることが必要になる。

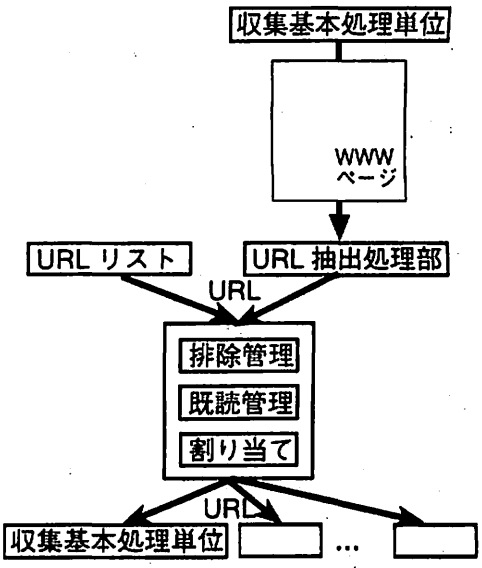

図 1: WWW ロボットにおける URL のフロー

WWW ロボットにおける URL のフローを図 1 に示す。URL の入力元は、起動時に与えられるリストか、収集されたページから URL を抽出する処理部かのどちらかである。出力先は、収集基本処理単位である。

入力から出力の間に必要とされる処理は以下の通りである。

- 割り当て: 当該 URL をどの収集基本処理単位に担当させるか
- 既読管理: 当該 URL がすでに収集されていないか
- 排除管理: 当該 URL の収集が許可されているか

割り当て処理では、同じ WWW サーバに複数のコネクションが張られないように、同じ WWW サーバに属する URL は単一の収集基本処理単位に割り当てることが重要である。既読管理は、すでに収集したページを再度サーバから取得することを防ぐ。排除管理は、収集が禁止されたページを収集しないようにするために必要となる。WWW ロボットの管理者が収集対象にしないよう定めた WWW サーバや URL に該当していないか、Robots Exclusion Protocol [6] で定められた方法で WWW サーバ管理者によって収集を禁じられていないかを判断する。

これらの処理を全て、特定のマシン/プロセスで集中的に行うこともできるが、そうすると受信コネクション多重度を上げていくにつれて、この処理がボトルネックになるのは明らかである。

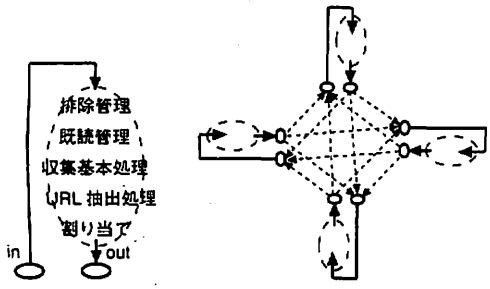

図 2: 静的割り当てを行う場合の処理単位とその相互関係

したがって、各収集基本処理単位がこの処理を行う方法として、静的割り当てを採用することにした。静的割り当てとは、例えば WWW サーバのホストネーム文字列のハッシュ値によって割り当てるといったように、WWW サーバが決まれば一意にそのサーバからの収集を担当する収集基本処理単位が定まるような方法である。この方法では、特定の WWW サーバに関する処理は最後まで特定の収集基本処理単位によって行われるため、排除管理および既読管理も収集基本処理単位ごとに行える。この各収集基本処理単位ごとに割り当て機能を持たせた場合の概念図を図 2 に示す。各収集基本単位間の関係はメッシュ型になる。

大規模な WWW ロボットでは、一般に排除管理や既読管理に利用されるデータベースの規模も大きくなるが、この方法ではこれらのデータベースを各収集基本処理単位ごとに分割でき、小さくできるというメリットもある。ただし、静的割り当てには、割り当て方法によっては URL が多く供給される収集基本処理単位と、少なく供給される単位とが生じ、単位ごとの稼働率にばらつきが出るという問題がある [7]。

5 実装について

この設計に基づき、我々は Solaris 2.6 を OS とするパーソナルコンピュータ (=PC) 上でプロトタイプの実装を行った。1 スレッド†で収集基本処理単位を 1 つ実現する。さらに、その収集基本処理単位が担当する WWW サーバについての既読管理、排除管理もそのスレッド内で行う。ただし、URL 割り当ての処理は、1 プロセスで利用できるファイル記述子に上限 (Solaris 2.6 では 1024) があるため、プロセスごとにまとめて行うことにした。

1 スレッドあたり、WWW サーバからのデータの受信, 既読管理, 排除管理で合計 3 つのファイル記述子を必要とする。1 プロセスあたりのスレッド数を t、総プロセス数を p、多重度を M、1 プロセスあたりのファイル記述子の上限を f_{max} とする。

各スレッドが互いに直接通信する場合、「他のスレッドとの通信に利用される記述子の数」と「その他に利用される記述子の数」の和が f_{max} 以下でなければならない。議論を単純化するために自スレッドへの URL 供給にもファイル記述子を使って通信すると仮定する。この場合、

$$f_{max} \geq pt^2 + 3t$$

また、定義より

$$M = t \times p$$

であるが、この 2 式において $f_{max} = 1024$ とした時、$t = 1$ の時 M が最大値 1021 を取る。この場合 $p = 1$ となり、1 プロセス内のスレッドは 1 になってしまう。

一方、プロセス内のスレッドが収集したページから抽出された URL を、プロセスごとに配分することを考える。ファイル記述子の上限から生じる制約条件から

$$f_{max} \geq 3t + p$$

また、定義より

$$M = t \times p$$

であるが、この 2 式において $f_{max} = 1024$ とすると、$t = 170, p = 514$ の時 M が最大値 87380 を取る。したがって、プロセスごとに URL の配分処理を行うことで、スレッドごとに配分処理するより十分な多重化が実現できることがわかる。

プロセス間の URL のやり取りについてはファイルを利用した。URL の出力元プロセスと入力先プロセスが決まると、そのやり取りに対応するディレクトリが定まるようにし、そのディレクトリに 0 から順に番号のついたファイルが生成する

†Solaris 2.6 の提供する POSIX スレッドライブラリを利用

ようにする。出力元プロセスは、一定数の URL をファイルに出力するとそのファイルをクローズし、次は 1 だけ番号の大きなファイルを開いて URL を出力する。これによって、この URL を受け取るプロセスは、最大の番号がついたファイル以外を順に読んでいくことで、出力元の書き出しと競合せずに整合性を保って URL を受信することができる。この方法と NFS を利用することにより、プロセスが同一のマシンに存在しているか否かを意識せず、URL 配分を相互に行える。

6 評価

このプロトタイプを用い、下記のように条件を変えながら実験を行った。

- 1 プロセス内のスレッド数を変化させて収集速度を調べる。
- スレッド数を固定し、1 マシン内で実行するプロセス数を変化させて収集速度を調べる。
- 1 マシン内のプロセス数、1 プロセス内のスレッド数を固定し、マシン数を変化させて収集速度を調べる。

それぞれの収集実験は 3 時間ずつ行われ、各回の収集件数を比較した。訪問間隔は 5 秒に固定した。収集に利用できるネットワーク接続の帯域幅は 100 Mbps である。PC 上ではそれぞれネームサーバを cache-only サーバの設定で動作させ、PC 上での DNS 解決はこのネームサーバを介して行った。また、各実験を行う前には、ネームサーバ内の cache エントリを消去するためにネームサーバの再起動を行った。

複数のプロセスを利用する実験では、最初に収集すべき URL は、収集前に各プロセス毎に分配しておく。このリストが十分に大きければ、各プロセスは、他のプロセスが発見した URL を読み込む必要がないので、プロセス間協調コストは発生しないとみなせる。本論文では、提案するアーキテクチャならば、収集処理単位を増やしても協調コストはあまり増えず、収集性能が向上するという仮説の検証を目指すため、意図的に最初に与える URL 数を小さく制限し、協調が起こるようにした。

しかし、最初に与える URL の数が極端に小さいと収集中に発見される URL が少なくなり、再帰的な収集が継続的に行われないことがある。そこで、(スレッド数 × 100) 個の URL をそれぞれのプロセスに与えて収集を開始させた。予備実験の結果、この程度の URL を与えると、最初に与えた URL 以外に新規に発見した URL が実験時間中に継続的に収集されることがわかっている。

利用した PC の仕様を表 3 に示す。

表 3: 評価実験に利用した PC

CPU	PentiumII 450MHz
RAM	640MB
HDD	18GB x 2
Network	100Base-TX
OS	Solaris 2.6 (for Intel)

6.1 スレッド数に関するスケーラビリティ

1 プロセス内のスレッドの数を 50, 100, 150 にそれぞれ変化させたところ、図 4 のように収集件数が変化した。現在の実装では、統計記録などを出力するために、1 スレッドあたり 4 以上のファイル記述子を利用しており、スレッド数を 200 にして実行することはできなかった。

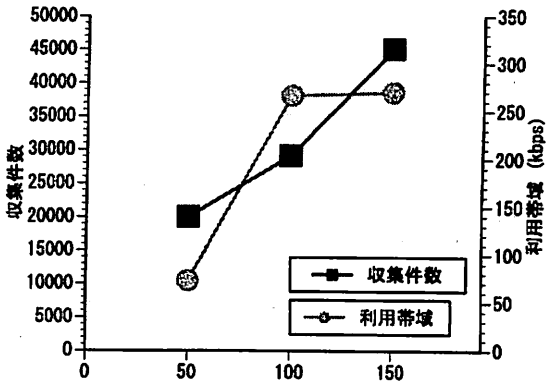

図 4: スレッド数に関するスケーラビリティ

このグラフが示すように、スレッド数を増加させることで、収集速度が向上している。

6.2 プロセス数に関するスケーラビリティ

利用する PC を 1 台に限定し、1 プロセスあたりのスレッド数を上記実験で最も性能の良かった 150 に固定し、収集に利用するプロセス数を 1 から 3 まで変化させた。この際の、各プロセス数での収集件数と、収集で利用した帯域幅を図 5 に示す。

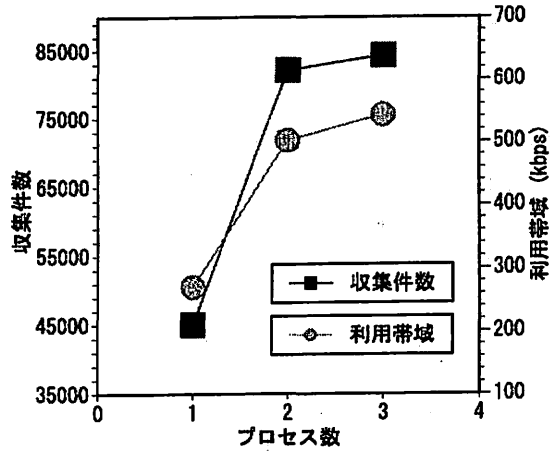

図 5: プロセス数に関するスケーラビリティ

このグラフが示すように、1 台の PC 上で利用するプロセス数を増加させることで、利用する帯域幅の増加にほぼ比例して、収集速度も向上している。また、実験に利用した PC では、測定の結果、3 プロセスを実行しても CPU がボトルネックになるようなことはなかった。

6.3 マシン数に関するスケーラビリティ

1 台の PC 上で実行するプロセスの条件を「150 スレッドのプロセス」「1 PC 上で同時に 1 プロセス」に固定し、利用する PC の台数を 1 台から 5 台まで増加させた際の、収集件数と収集で利用した帯域幅の変化を図 6 に示す。

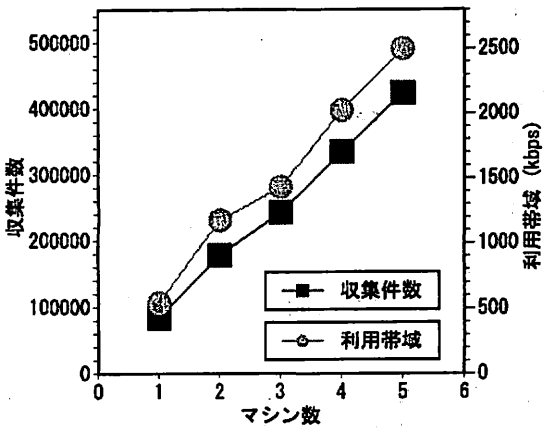

図 6: マシン数に関するスケーラビリティ

このグラフが示すように、PC の台数を増加させると、利用する帯域幅の増加にほぼ比例して、収集速度が向上している。

7 考察

本論文では、WWW ロボットの収集速度向上のために、収集の多重度を向上することを追求した。多重度の向上には、1 プロセス内のスレッド数の増加、1 マシン内のプロセス数の増加、利用するマシン数の増加が考えられるが、我々の設計ではこの全ての方法が収集速度向上に結び付くことを目指した。この設計に基づいたプロトタイプを利用した実験では、この全ての方法が収集速度向上に対して有効であることが明らかになった。

このプロトタイプを 5 台までの PC で利用する際、収集速度がスケールすることが示されたが、今後は何台まで速度がスケールするかを明らかにしていく。

参考文献

[1] Brian Pinkerton, "Finding What People Want: Experiences with the WebCrawler", Proceedings of the First WWW Conference, 1994.

[2] Sergey Brin and Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", Proceedings of the Seventh WWW Conference, 1998.

[3] 能登 信晴, 竹野 浩, 小橋 喜嗣, "インターネット検索サービスのための分散型情報収集", マルチメディア, 分散, 協調とモーバイル (DICOMO'98) シンポジウム論文集, 1998.

[4] 山名 早人 他, "分散型 WWW ロボットによる WWW 情報収集", 第 9 回データ工学ワークショップ (DEWS'98) 論文集, 1998.

[5] 亀井 聡, 河野 浩之, 長谷川 利治, "分散型 Web ロボット構築のための性能評価", 第 9 回データ工学ワークショップ (DEWS'98) 論文集, 1998.

[6] Martijn Koster, "Robots Exclusion", http://info.webcrawler.com/mak/projects/robots/exclusion.html

[7] 竹野 浩, 能登 信晴, "情報収集ロボットの収集特性の解析", 情報処理学会第 61 回全国大会論文集 (3), 2000.