情報処理学会論文誌　データベース　Vol.6　No.2　85–97 (Mar. 2013)

Producer-Consumer型モジュールで構成された並列分散Webクローラの開発

上田 高徳[1,a)]　佐藤 亘[1]　鈴木 大地[1]　打田 研二[1]　森本 浩介[1]　秋岡 明香[1]
山名 早人[1,2]

受付日 2012年9月20日, 採録日 2013年1月6日

概要：Webクローラは，クローリング済みURLの検出やWebサーバに対する連続アクセス防止といった処理を実行しながらデータ収集を行う必要がある．Web空間に存在する大量のURLに対して高速な収集を実現するために並列分散クローリングが求められるが，省資源でのクローリングを行うためにも，処理の時間計算量と空間計算量の削減に加え，計算機間の負荷分散も必要である．本論文で提案するWebクローラは，クローリング処理をProducer-Consumer型のモジュール群で実行することにより，これまでの被クロールWebサイト単位での負荷分散でなく，Webクローラを構成するモジュール単位での負荷分散を実現する．つまり，Webクローラを構成する各モジュールが必要とする計算機資源に応じた分散処理が可能になり，計算機間での計算負荷やメモリ使用量の偏りを改善することができる．また，ホスト名やURLを管理するモジュールは時間計算量と空間計算量に優れたデータ構造を利用して構成されており，大規模なクローリングが省資源で可能になる．

キーワード：Webクローラ，並列分散処理，Producer-Consumerモデル

A Parallel Distributed Web Crawler Consisting of Producer-Consumer Modules

Takanori Ueda[1,a)]　Koh Satoh[1]　Daichi Suzuki[1]　Kenji Uchida[1]
Kousuke Morimoto[1]　Sayaka Akioka[1]　Hayato Yamana[1,2]

Received: September 20, 2012, Accepted: January 6, 2013

Abstract: Web crawlers must collect Web data while performing tasks such as detecting crawled URLs and preventing consecutive accesses to a particular Web server. Parallel-distributed crawling is carried out at a high speed for the enormous number of URLs existing on the Web. However, in order to crawl efficiently, a crawler must realize load balancing between computers in addition to reducing time and space complexities in the crawling process. The Web crawler proposed in this paper crawls the Web using producer-consumer modules, which compose the crawler, and it realizes load balancing per module and not per crawled Web site. In other words, it realizes load balancing that is appropriate to certain computer resources necessary for the modules that compose the Web crawler; in this way, it improves biases in computation loads and memory utilization between computers. Moreover, the crawler is able to crawl the Web on a large scale while conserving resources, because the modules that manage host names or URLs are implemented by data structures that are temporally and spatially efficient.

Keywords: Web crawler, parallel distributed computing, Producer-Consumer model

1. はじめに

Web上のデータを用いた研究やサービス，ビジネスを行うにあたり，Webを巡回してデータを収集するWebクローラは必要不可欠な存在である．Webクローラはクロー

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 169–8555, Japan
[2] 国立情報学研究所
National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[a)] t-ueda@fuji.waseda.jp

リング起点として与えられたURL集合からWebページの取得を開始し，ハイパーリンクに従ってWebページや動画像といったWebデータを収集していく．クローリング速度は速い方が望ましいが，同一Webサーバに対して短い間隔でアクセスして，負荷を掛けすぎることがあってはならない．2012年の報告[2]によると，あるサイトに対するアクセス数の6.68%をWebクローラが占めるが，人間とは異なるアクセスパターンが原因で，サーバ負荷の31.76%をWebクローラが発生させると報告されており，マナーを守った高速クローリングが必要といえる．

しかし，Web空間には2008年の段階で1兆個のURLが存在すると報告されており[*1]，大量のURLを管理しながら，高速収集とWebサーバに対する負荷低減を実現する必要がある．より多くのWebページを収集するために，並列分散クローリングが必要になるが，できる限り少ない計算機でのクローリングを実現するためにも，クローリング処理の時間計算量と空間計算量を削減したうえで，計算機間の負荷分散を行い，スケーラブルなクローリングを実現する必要がある．これら，マナーを守ったクローリング方法，スケーラブルな並列分散実行方法，省資源でのクローリング方法は，すべて並列分散クローラに重要な要素である．特に並列分散実行方法は，アルゴリズムやデータ構造と密接に関わるため，Webクローラ全体の仕組みを総合的に議論する必要がある．これまで商用Webクローラは成功を収めているが，内部構成について十分に公開されているとはいえず，並列分散Webクローラの構成方法について論文で議論することは価値がある．

我々は，2003年度～2007年度に実施されたe-Societyプロジェクト[*2]において100億ページ以上を収集[19]した．そこで得られた様々な経験から，より高性能な並列分散Webクローラを実現するために，2008年度より新しいWebクローラの開発に取り組んできた．これまでに，マナーを守ったクローリングをO(1)の時間計算量で実現するクローリングスケジューラ[20]や，URLの重複除去を小さな計算空間で行う手法[18]，リアルタイムなデータ提供が可能でカスタマイズ性に優れたWebクローラの構成方法[17]などを提案してきたが，並列分散クローリングについては議論してこなかった．

並列分散クローラでは処理の並列分散化に加え，負荷分散も解決する必要がある．分散クローラに関する既存文献[10], [12]では，計算機ごとにクローラを立ち上げ，クローリング処理をWebサイト単位で分割して計算機に割り当てている．容易に分散クローリングが可能になるが，Webサイトが保持するURL数には偏りがあるため，たとえばTwitterのような大量のURLを持つWebサイトが割り当てられた計算機は過負荷になる．我々がe-Societyプロジェクトで収集を試みた際も，Webサイト単位の分割方法を用いたところ計算負荷の偏りが生じて問題となった．より細かい処理単位で並列分散実行できるようクローラを構成し，負荷分散を図る必要がある．

そこで本論文では，我々が開発したProducer-Consumer型モジュールで構成された並列分散Webクローラについて述べる．本クローラは，我々がこれまで提案した手法をProducer-Consumer型モジュール群で実装したもので，我々が開発している並列分散処理フレームワークのQueueLinker[16]上で動作する．同一IPアドレスを持つWebサーバに対するアクセス間隔が指定時間以上になることを保証し，マナーを守ったクローリングが可能である．モジュールのそれぞれは，任意のスレッド数かつ任意の計算機台数で並列分散実行することができる．このため，クローラの処理単位ごとに計算資源の割当てが行える．ホスト名やURLを管理するモジュールは時間計算量と空間計算量に優れたデータ構造を利用して構成されており，大規模なクローリングが省資源で可能になる．また，Producer-Consumerモデルを採用することで，モジュール間を流れるデータを利用すれば，研究目的に沿って必要なデータをリアルタイム解析したり，必要なデータのみを保存したりといったカスタマイズを容易に行える点が副次的な効果として生まれる．

本論文は以下の構成をとる．2章でWebクローラに要求される項目と既存クローラについて整理し，3章で我々のクローラの位置付けを述べる．4章で提案クローラの実行モデルを説明し，5章で開発したWebクローラのモジュール実装について述べる．6章において評価実験について述べ，7章でまとめる．

2. Webクローラの要求仕様と既存クローラ

本章ではWebクローラに求められる仕様について議論する．これまでのクローラに関する主要文献[6], [10], [12], [14]のうち，すべての文献で共通して検討されている事項は，スケーラビリティとクローリングマナーである．Webに存在する大量のデータに対応するため，特に分散クローリングを行う際には計算機台数に対してスケールできる必要がある．そのうえで，Webサーバに対するマナーを守りながらクローリングする必要がある．また，文献によってはクローリング済みのページの再収集や，計算機の故障に対する耐障害性，動作環境を問わないこと，設定の容易さ，目的に合わせた機能追加の容易さ，といった点も議論している．これらは本論文では詳細を議論しないが，本クローラの機能追加で実現できると考えている．

したがって，本論文では主にマナーを守ったクローリングとスケーラビリティの観点から我々のクローラについて

*1 “We knew the web was big...,” 入手先 ⟨http://googleblog.blogspot.jp/2008/07/we-knew-web-was-big.html⟩

*2 「文部科学省リーディングプロジェクトe-Society基盤ソフトウェアの総合開発」, 入手先 ⟨http://cif.iis.u-tokyo.ac.jp/e-society/⟩

説明する．両項目を細分化し，以下の各項目の必要性を本章では説明する．

(1) マナーを守ったクローリングに必要な項目
  (a) 同一 Web サーバに対するアクセス間隔を十分にとること
  (b) robots.txt および meta タグ内の指示に従ってクローリングを行うこと
  (c) 同一ページを頻繁にアクセスしないこと
  (d) 外部 DNS サーバに対する負荷を軽減すること
  (e) エラー発生時に適切に対応すること
(2) スケーラビリティに必要な項目
  (a) 計算機の負荷分散ができ，高速に並列分散クローリングできること
  (b) 省資源でクローリングできること
  (c) 重要なページを優先的に収集できること

2.1　マナーを守ったクローリング

ある Web ページ内のリンク先は同一 Web サーバであることがほとんどであるから，間隔をあけずにリンク先をクローリングするとサーバに対する負荷を高めてしまう．前述したように，サーバに対する負荷の 31.76%がクローラによるものという報告 [2] もあり，アクセス間隔を十分にとったマナーあるクローリングが必要である．

また，各サーバはルートディレクトリに robots.txt [*3] を配置して，クローリング対象のページやクローリング間隔を制御することができる．robots.txt では，クローラごとにアクセス許可や不許可のページを設定できる．これまでの報告 [13], [15] を参考にすると，3 割から 4 割程度の Web サーバに robots.txt が設置されていると考えられ，robots.txt を解釈してクローリングすることは必須である．さらに，robots.txt では Crawl-delay を設定することでアクセス間隔を指定することができる．Crawl-delay は商用クローラでも守られていないこともある [2] が，商用クローラよりも認知度の低い本論文のクローラの場合は，Crawl-delay を守ってクローリングした方がサイト管理者に受け入れられやすいと考えられる．

そして，同一ページに連続して複数回アクセスすると Web サーバ負荷の増加になるだけでなく，クローリングの効率が悪くなるという問題がある．Web に存在する大量の URL に対して，クローリング済みの URL をいかに管理するかが課題となる．また，DNS サーバに対する負荷や名前解決によるオーバヘッドを軽減するために，1 度ホスト名の名前解決を行ったあとは一定時間キャッシュすることが望ましい．この場合も，Web に存在する大量のホスト名と IP アドレスの対応をいかに保持するかが課題となる．エラーが発生したページに対してアクセスし続けないよう，エラー発生時のリトライ回数の制限も必要である．

[*3] “The Web Robots Pages,” 入手先 ⟨http://www.robotstxt.org/⟩

2.2　スケーラビリティ

前述のようにマナーを守って高速かつスケーラビリティのあるクローリングを行う方法は，データ構造の観点からもチャレンジングな課題である．データ構造をメモリとストレージのどちらに，あるいは双方に格納するのかも検討する必要がある．IRLbot 以前のストレージベースの手法はストレージアクセスがオーバヘッドになることが指摘されており [6]，IRLbot では DRUM という効率の良いストレージベースのバッチ処理方法を提案している．また，物理メモリのみ用いる方法はスケーラビリティの確保が容易になるが，メモリ不足の際にはクローリング待ち URL や訪問済み URL などのデータを破棄する必要がある．

そして，効率的に Web データを収集するためには並列に HTTP コネクションを試みなければならない．レスポンスタイムとスループットは Web サーバごとに異なり，速度の遅い Web サーバも多いため，複数のスレッドでダウンロードを実行しなければ，十分な合計スループットを出すことができない．また，高速ダウンロード時には，HTML のパースやクローリング済み URL の検出といった処理に多くの CPU 時間が必要になり，並列分散化が必要となる．

分散時に重要になってくるのは計算資源の割当ての柔軟性である．従来の分散クローラは Web サイトごとに計算機を割り当てて処理を行っている．容易に分散クローリングが可能になるが，Web サイトが保持する URL 数には偏りがあるため，たとえば Twitter のような大量の URL を持つ Web サイトが割り当てられた計算機は過負荷になる．我々が 100 億ページを収集した経験 [19] でも，Web サイトごとの分散方法では計算機負荷の偏りが生じて問題になった．

さらに Web クローラの性能は，いかに高品質なページを収集できるかという点でも評価される．インターネット上には多くのスパムページが存在するため，それらのページによるトラフィックで重要なページのダウンロードが阻害される恐れがある．

2.3　既存クローラの収集速度

Web クローラを開発する試みは学術界でも長く続いており，少数ながら文献が継続的に発表されてきている．1999 年の Mercator [1] は単一計算機によるクローリングを行っており，毎秒 112 ページのダウンロードを行っている．Mercator に関する 2001 年のレポート [10] では，4 台の計算機による分散クローリングが行われている．以降，分散化の試みもさかんになり，2002 年の Ubicrawler [12] は日に 1,000 万ページ（毎秒換算で約 116 ページ），同年の他の論文 [14] では，2 台の計算機を利用して毎秒 140 ページなどとなっている．IRLbot [6] は単一計算機で毎秒

平均 1,789 ページの収集を達成している．Stanford 大学の WebBase [7] では 40 プロセスを用いて毎秒 6,000 ページ以上の性能を達成できるとしている．

3. 我々の Web クローラの位置付け

前述のように，既存の分散クローラに関する文献 [10], [12] では，クローリング処理を Web サイト単位で分割して計算機に割り当てている．クローラの機能ごとに使用するリソース量は異なるため，クローラの全機能を計算機の台数分複製するのではなく，より細かい機能粒度でリソース割当てを行えるようにクローラを構成するべきである．また，初期のクローラは毎秒数千ページのダウンロード速度における並列分散クローリングの評価ができていない．IRLbot のような高速クローラでは機能が密結合になっており，クローラの機能拡張が容易でないと考えられる．

これらの課題と前章で述べた要求を解決するために，本論文で示す並列分散クローラは，すべてのクローリングの機能を Producer-Consumer モジュールで構成した．**表 1** に 2 章で述べた要求仕様への対応方法を示した．各モジュールは，ホスト名，URL，IP アドレスのいずれかのハッシュ値に基づいて担当するデータ空間を分割でき，任意のスレッド数，かつ任意の計算機台数で動作することができる．データ構造はインメモリで保持し，異なるモジュールやインスタンス間では内部状態を共有しない．このことから，クローラの処理単位ごとに柔軟な計算資源の割当てを行うことができ，サイトごとにクローリング範囲を分割するよりも細粒度で計算機リソース割当てが可能となる．

図 1 を用いて本クローラの負荷分散の利点を説明する．いま，サイト名に対応する IP アドレスをキャッシュする DNS キャッシュと，URL の重複除去処理を考える．クローリング範囲を Web サイトごとに計算機に割り当てると，サイトが保持する URL 数の偏りにより，図 1 の (1a) と (1b) の計算機のように処理を担当する URL 数に差が生じる．DNS キャッシュはサイト単位の処理であるため，(1a) と (1b) で使用メモリ量は均衡するが，重複除去は URL 単位の処理であるため (1a) と (1b) では使用メモリ量に偏りが生じる．本クローラは，サイトに対する処理と URL に対する処理で別々に計算機を割り当てることが可能なため，図 1 中の (2a) と (2b) の計算機のように，負荷均衡化が可能になる．

さらに，限られたメモリ空間で可能な限りの情報を保持するために，BloomFilter [4] や Hat-trie [11] といった確率的データ構造やトライ構造を利用する．また，Producer-Consumer モデルを採用することで，収集データをデータストリームとしてユーザに提供することが可能になり，文献 [9] のようにアプリケーションのバックエンドとして利用することも可能である．我々が開発している QueueLinker のサポートにより，新しい機能を持つモジュールの追加や

表 1　要求仕様と本クローラの解決方法
Table 1　Our solutions for the requirements of a crawler.

項目	解決方法
(1a) アクセス間隔	最短アクセス間隔の下限を保証
(1b) robots.txt，meta タグ	対応
(1c) 同一 URL 判定	LRU と Bloom Filter を利用
(1d) DNS 負荷削減	Hat-trie によるキャッシュ
(1e) エラー対応	再アクセスしない
(2a) 並列分散クローリング	QueueLinker によるサポート
(2b) 省メモリ	Bloom Filter や Hat-trie の利用
(2c) 優先度付き収集	スケジューラの性質により実現

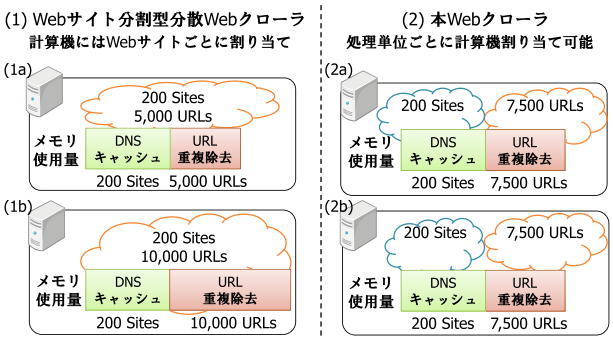


図 1　処理単位の分散による負荷均衡化の効果
Fig. 1　The benefit of load balancing on function level.

モジュール間の接続変更を容易に行うことができる．

このほか，既存論文では，クローリング済みページの再収集，計算機の故障に対する耐障害性，動作環境を問わないポータビリティ性，設定の容易さといった運用上重要な点も議論されている．クローリング済みのページの再収集については本論文で議論しないが，既存手法を本クローラに実装することが可能と考えている．耐障害性は我々の開発している並列分散処理フレームワークの QueueLinker 側の責任であるため本論文では扱わないが，QueueLinker も本クローラも Java で実装されておりポータビリティがある．設定の容易さも本論文で扱わないが，実装の拡張で対応できると考えられる．

4. 提案 Web クローラの実行モデル

本章では提案クローラの実行モデルについて述べる．実行ランタイムには我々が開発中の並列分散処理フレームワークである QueueLinker を用いる．本章では QueueLinker を用いた Web クローラの実行モデルについて詳述し，各モジュールの実装については 5 章で詳しく説明する．

4.1 Producer-Consumer モデルと QueueLinker

Producer-Consumer モデルはマルチスレッド処理のためのデザインパターンの一種である．QueueLinker では，Producer-Consumer の処理単位をモジュールと呼ぶ．1 つのモジュールは，入力されたデータに基づいて何らかの

処理を行い出力を生成する．異なるモジュールどうしは内部状態を共有せずに，キューを介してのみ通信する．Producer-Consumer モデルを用いるとモジュールごとにスレッド数を調整することができる．また，異なるモジュールが同一計算機で動作する必要がない場合，キューの入出力を計算機間で転送すれば並列分散実行も可能になる．

QueueLinker は Producer-Consumer モデルで実装されたモジュールと，モジュール間の接続関係を受け取り，インスタンス生成やモジュール間のデータの転送を自動的に行って並列分散実行を実現する．QueueLinker を用いることでクローラのコード内には計算機間やモジュール間のデータ転送の仕組みを含む必要はなく，簡潔な記述でクローラを実現することができる．以下，本クローラを例に，QueueLinker を用いたモジュールの実装方法と並列分散実行モデルについて説明する．

4.2　データ構造とモジュール構造

まず，本クローラのモジュール間の通信に利用するデータ構造とモジュールの構造について説明する．**表 2** にデータ構造を示した．以下，このデータ構造を CrawlingData と呼ぶ．処理対象の URL 1 つに CrawlingData のインスタンス 1 つが対応する．各モジュールは CrawlingData を入力として受け取り処理を実行したあと，必要に応じて CrawlingData を更新して出力する Producer-Consumer 型の実装とする．たとえば，ダウンロード処理を行うモジュールは，CrawlingData を 1 つ受け取ると url に格納されている URL にアクセスし，ダウンロードしたデータを downloadedData に格納する．そして，更新済みの CrawlingData を出力する．

List 1 に，入力キューを N_{in} 個と出力キューを 1 個持つモジュールの疑似コードを示した．データが入力されたキューは *id* で識別できる．入力キューに応じた処理を実行したあと，必要に応じて処理結果を CrawlingData に格納して出力する．なお，null を返した場合データは破棄される．したがって，クローリング済み URL や不正な URL に対するフィルタを行うモジュールは，フィルタ対象の URL の場合は null を返す．List 1 には明記していないが，モジュールは変数やトライ木，BloomFilter など処理に必要なデータ構造を内部状態として持つことができる．

4.3　並列分散実行モデル

以上のデータ構造とモジュール構造をもとに，いかに並列分散実行を実現するか述べる．**図 2** は，「HTML 解析」と「重複除去」の 2 つのモジュールの実行方法のパターンについて示している．破線で囲われた領域が 1 台の計算機に相当する．(1) は 1 つのモジュールごとに 1 インスタンス生成し，1 スレッドを割り当てて実行している場合である．ここで並列実行を行うことを考える．一般的な

表 2　CrawlingData：モジュール間の通信に用いるデータ構造(抜粋)
Table 2　CrawlingData: Main members of the data structure for communication between the modules.

型	変数名	役割
String	url	処理対象の URL
byte[ ]	ipAddress	解決済み IP アドレス
byte[ ]	downloadedData	ダウンロードしたデータ
int	robotsFlag	robots.txt の有無 (0：無，1：有，2：不明)
boolean	downloadable	robots.txt による DL 可否 (true：許可，false：禁止)
int	crawlDelay	Crawl-delay の設定値． 設定なしの場合は 5 秒とする
Exception	exception	例外が発生した場合に保持

List 1 Module Example

```
 1: procedure MODULE(data : CrawlingData, id : int)
 2:     if id = 0 then
 3:         data に対して処理 0 を実行
 4:     else if id = 1 then
 5:         data に対して処理 1 を実行
 6:   (... 中略...)
 7:     else if id = N_in − 1 then
 8:         data に対して処理 N_in − 1 を実行
 9:     end if
10:     return data
11: end procedure
```

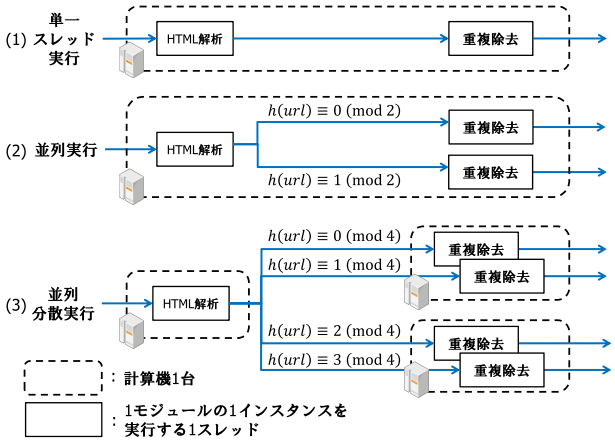


図 2　並列分散実行のモデル
Fig. 2　The model of parallel distributed execution.

Producer-Consumer モデルのマルチスレッド実行では，1 つのインスタンスを複数のスレッドで並列に実行する．しかし，内部状態を持つモジュールの場合はスレッドセーフな実装にする必要があるうえに，並列度が上がったときにはモジュール内の排他制御によるオーバヘッドも問題になりうる．

一方，図 2 中の (2) のように 1 スレッドに 1 インスタンスずつ用いてデータ並列で実行すれば，スレッドセーフな実装にする必要もなくなり，モジュール内の排他制御も不要になる．内部状態を持つモジュールの場合でも，データ

のハッシュ値に基づいて処理するインスタンスを決めることができれば，インスタンス間で内部状態を共有せずに並列実行が可能である．しかし，このようなデータ並列の実行方法が可能かは，クローリング処理をハッシュ分割できるかにかかっている．

本クローラは，すべてのモジュールがハッシュ分割可能なように構成されており，以下の 3 つのハッシュ関数のいずれかを利用して処理を振り分ける．

- $h(Host)$：ホスト名のハッシュ値を求める関数
  - ホストごとの情報のキャッシュと robots.txt の処理において利用
- $h(URL)$：URL のハッシュ値を求める関数
  - クローリング済み URL の管理において利用
- $h(IP)$：IP アドレスのハッシュ値を求める関数
  - 同一 Web サーバへのアクセス間隔をスケジューリングする処理において利用

たとえば，ホスト名に対応する IP アドレスをキャッシュするモジュールを考えると，同一ホストの URL を同一インスタンスが処理する限り，キャッシュに存在すれば必ずヒットする．また，クローリング済み URL を除去するモジュールを考えると，同一 URL を同一インスタンスが処理する限り必ず重複除去できる．これらの処理は，必ず 1 インスタンスのみで完結し，他のインスタンスの内部状態にアクセスする必要はない．したがって，ハッシュ分割することにより，任意の並列数で実行が可能になる．

同様の方法で分散実行も容易に実現できる．すべてのモジュールは内部状態を共有せず CrawlingData のみで通信する．したがって，計算機間で CrawlingData をシリアライズして転送すれば分散実行も可能になる．図 2 中の (3) に並列分散実行をしている場合を示した．

以上により，各モジュールは任意のスレッド数・任意の計算機数で実行することが可能になる．計算機台数を増やすことで，1 台の計算機あたりの計算負荷，およびメモリ使用量を任意に削減することができる．これら，ハッシュ値による並列分散処理や計算機間のデータ転送は QueueLinker が自動的に行う．

4.4　仮想モジュールと Switcher

QueueLinker はこのほかに，モジュールの出力内容に応じて次に転送するべきモジュールを決定する Switcher と呼ばれる機構と，同一のモジュールを異なるフロー内で再利用するための仮想モジュールと呼ばれる機構を備えている．

図 3 を用いて，これら 2 つの機能を説明する．Switcher は図中の数字を含む円で表現されており，円の中の数字は分岐数を示す．仮想モジュールは破線の矩形で示されている．Switcher の疑似コードを List 2 に示した．転送先を指示する値を返すことで，QueueLinker が適切な転送先にデータを転送する．図の例で A の出力は，Switcher が 0 を返すと B に，1 を返すと C に転送される．B の出力は A に再び入力されるが，この入力に対する A の出力は D に入力される．すなわち，左側の A と右側の A では出力の転送先モジュールが異なる．このような A を仮想モジュールと呼び，入出力のルートは異なるものの，ともに同一のインスタンスで実行される．仮想モジュールはフロー内でのモジュール再利用を実現するものである．最後に，D の出力は E と F に複製されて入力される．複製も QueueLinker が自動的に生成する．

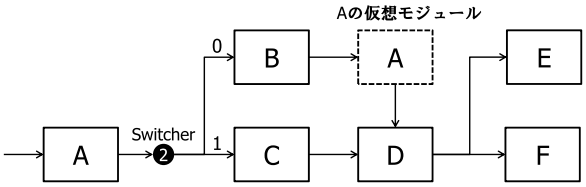


図 3　Switcher と仮想モジュール
Fig. 3　A switcher and a virtual module.

List 2 Switcher Example

```
1: procedure SWITCHER(data)
2:     return f(data) ▷ data の内容に応じて転送先を整数で返す
3: end procedure
```

5.　提案 Web クローラのモジュール実装

本章では我々が開発した並列分散 Web クローラを構成する個々のモジュールの詳細と処理アルゴリズムについて述べる．4 章で説明した実行モデルを用いて実装されており，**表 3** に示したモジュール群が**図 4** のように接続されて動作する．図 4 中の矩形が 1 つのモジュールに相当し，複数回出現しているモジュールは仮想モジュールである．入力キューを複数もつモジュールについては入力キューを併記している．

本クローラでは，我々が以前に提案した時間計算量が O(1) のクローリングスケジューラ [20] を単純化し，並列分散実行可能にして利用する．本スケジューラは同一 IP アドレスに対するアクセスが指定された時間間隔以上になることを保証する．再クローリングには対応しないが，PageRank の総和で比較すると，同じダウンロードページ数であれば幅優先よりも効率良く収集できる [20]．また，robots.txt と meta タグの解釈を行い，DNS への問合せ結果はキャッシュする．従来提案した URL の効率的な重複除去手法 [18] をオンメモリ処理のみに変更し，クローリング中に出現する URL を効率良く検出する．大量の URL やホスト名を管理する必要がある，(h) Duplicated URL Checker，(i) Host Data Cache の 2 つのモジュールは時間計算量と空間計算量に優れたデータ構造を利用して構成されており，大規模なクローリングが省資源で可能になる．

表 3　クローラを構成するモジュール一覧
Table 3　List of modules which compose our crawler.

モジュール名	分散戦略	入力キュー数
(a) Scheduler	IP	2
(b) Scheduler Timer	Free	1
(c) robots.txt Downloader	Free	1
(d) Downloader	Free	1
(e) HTML Parser	Free	1
(f) URL Format Filter	Free	1
(g) Explicit URL Filter	Free	1
(h) Duplicated URL Checker	URL	1
(i) Host Data Cache	Host	4
(j) Domain Name Resolver	Free	1
(k) robots.txt Processor	Host	2
(l) Data Store	Free	1
(m) Seeder	Single	0

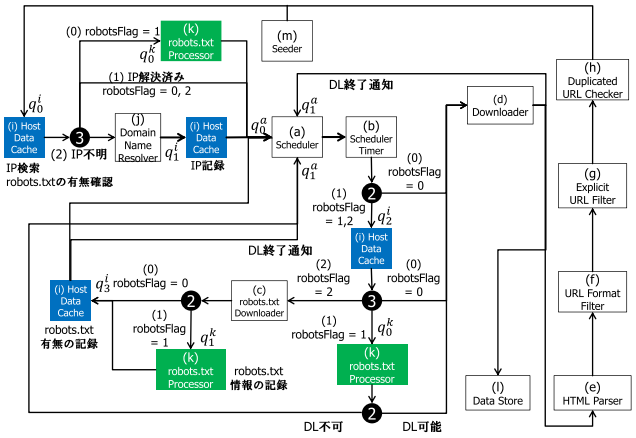


図 4　モジュールの接続関係
Fig. 4　The data connections between modules.

5.1　各モジュールの概要と分散戦略

本クローラを構成するモジュールは，内部状態を保持するモジュールと保持しないモジュールに分かれる．内部状態を持たないモジュールは表 3 の分散戦略の項目が Free になっている．これらの内部状態を持たないモジュールはどの計算機のどのスレッドで実行しても問題ない．たとえば Web ページをダウンロードする処理やそのパース処理は内部状態を持たず，任意の計算機で処理を実行することができる．

次に，内部状態を持つモジュールは表 3 の分散戦略の項目が IP，Host，URL のいずれかで示してある．4 章で説明したように，これらのモジュールの並列分散処理においては各データのハッシュ値によってハッシュ分割する．たとえば Scheduler は IP アドレスごとにクローリング待機リストを管理するので，$h(IP)$ の値で処理の担当範囲を分割すればよい．また，Host Data Cache はホスト名ごとに対応を管理するので，$h(Host)$ の値で分割すれば良い．robots.txt Processor も同様である．Duplicated URL Checker は URL を対象とする処理であるから，$h(URL)$ の値で分散させれば良い．

なお，Seeder に関しては複数実行されると重複した URL が Scheduler に入力されてしまうため，ただ 1 つのスレッドだけで実行される必要があり，表 3 では Single と表記している．

5.2　各モジュールの説明

本節では，クローラの機能ごとに分類してモジュールを説明していく．実装が自明でないモジュールに関しては疑似コードをともに示す．

5.2.1　クローリングスケジューラ

クローリングスケジューラは本クローラの最も重要な機構といえる．本スケジューラは，同一 IP アドレスを持つサーバへのアクセス間隔を指定時間以上にすることを O(1) の時間計算量で保証することができる．これは我々が過去に提案したスケジューラ [20] をより単純化したうえで，robots.txt の Crawl-delay に対応させ，4 章で説明したハッシュ分割方法により並列分散実行できるようにしたものである．表 3 にある (a) Scheduler と (b) Scheduler Timer という 2 つのモジュールでこの機能を実現している．本論文では同一 IP アドレスに対する最低アクセス間隔は 5 秒として説明する．

アルゴリズムを Algorithm 1 に示した．Scheduler は 2 つの入力キュー q_0^a，q_1^a を持つ．q_0^a にはクローリング中に新しく発見された URL に対応する CrawlingData が入力され，q_1^a にはダウンロードが完了，あるいはエラー終了した CrawlingData が入力される．また Scheduler は，**図 5** のように L 個のダウンロード待機 URL リスト $l_0, l_1, \ldots, l_{\mathrm{L}-1}$ を持つ．ある 1 つのリスト中には同一 IP アドレスの Web サーバの URL が 2 つ以上含まれないように管理し，並列にダウンロードが実行されるのはただ 1 つのリストに含まれる URL のみになるように制御する．1 つのリストの並列ダウンロードが完了して次のリストの並列ダウンロードを開始する前に 5 秒待つことで最低アクセス間隔を保証する．

Algorithm 1 では，q_0^a に入力された URL を，どのリストに追加するかテーブル $n_0, n_1, \ldots, n_{\mathrm{H}-1}$ を参照して決定する．URL のホスト名に対応する IP アドレスを整数と見なし，H による剰余値が i の場合，$l_{n_i \pmod{\mathrm{L}}}$ がその URL を追加するリストの候補である．異なる IP が同一剰余値になることはあるが，衝突を無視しても同一リストに同じ IP アドレスの URL が含まれることはない．ここでは，ダウンロード中のリストを l_c として $a_m \equiv c + m \pmod{\mathrm{L}}$ としたときに，$\{l_{a_m} | 1 \leq m \leq \mathrm{L}-1\}$ のリストのうち，m が小さいリストから URL を格納していく．ただし，robots.txt に Crawl-delay が設定されている場合は，指定時間空くように間隔をあけてリストに格納していく（Algorithm 1 の 15，21 行目）．Crawl-delay が設定されていない場合は，

Algorithm 1 (a) Scheduler

ダウンロード待機 URL リスト：$l_0, l_1, ..., l_{\mathrm{L}-1}$
整数値：$n_0, n_1, ..., n_{\mathrm{H}-1}$ (すべて 0 で初期化)
現在のステップ：c (-1 で初期化)
現在ダウンロード中の URL 数：p (0 で初期化)

```
 1: procedure SCHEDULER(data : CrawlingData, id : int)
 2:     if id = 0 then
 3:         NewURL(data)                          ▷ q_0^a への入力を処理
 4:     else if id = 1 then
 5:         p = p − 1                             ▷ q_1^a への入力を処理
 6:     end if
 7:     if p = 0 then
 8:         return GoNext()
 9:     end if
10: end procedure
11:
12: procedure NEWURL(data : CrawlingData)
13:     r ≡ data.ipAddress の符号なし 32 bit 整数表現 (modH)
14:     if n_r ≤ c then
15:         n_r = c + ⌈data.crawlDelay/5⌉
16:     end if
17:     if n_r ≥ c + L then
18:         return null                           ▷ URL は破棄
19:     end if
20:     i ≡ n_r(modL)
21:     n_r = n_r + ⌈data.crawlDelay/5⌉
22:     l_i = l_i ∪ data
23: end procedure
24:
25: procedure GONEXT
26:     m = 1
27:     while m ≤ L − 1 do
28:         c = c + 1
29:         i ≡ c(modL)
30:         if |l_i| ≠ 0 then
31:             l = l_i
32:             l_i = φ
33:             p = |l|
34:             return l
35:         end if
36:         sleep(5 seconds)                      ▷ 5 秒間スレッド停止
37:         m = m + 1
38:     end while
39:     return null        ▷ 未ダウンロードの URL が 1 つもない
40: end procedure
```

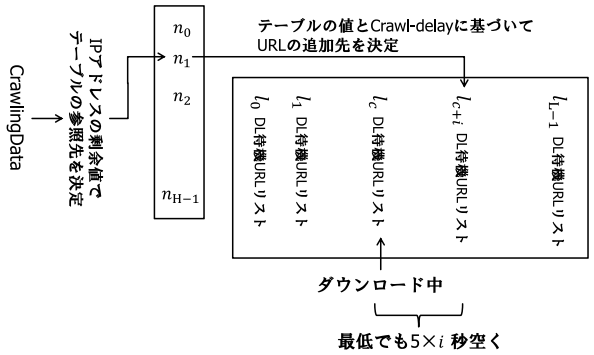


図 5　スケジューラの概要図

Fig. 5　The conceptual diagram of the scheduler.

Algorithm 2 (b) Scheduler Timer

```
1: procedure SCHEDULERTIMER(list:List<CrawlingData>)
2:     sleep(5 seconds)                          ▷ 5 秒間スレッド停止
3:     return list.toArray()    ▷ CrawlingData を 1 つずつ出力
4: end procedure
```

CrawlingData の crawlDelay に 5 秒が初期値として設定されているため連続してリストに格納されていく．リストが不足して格納することができない場合 URL は破棄する．重要なページであれば，クローリング中に再度出現することが期待できる [20]．

また，q_1^a に入力された CrawlingData の数を数えることで，ダウンロードが完了した URL 数を判定することができる．l_c 内の全 URL のダウンロードが完了したら次のリストを出力する．より正確には，$\{l_{a_m} | 1 \leq m \leq \mathrm{L}-1\}$ のリストのうち，有効な URL を持つ m が最も小さいリストのダウンロードが実行されるよう出力する．そして，Scheduler の次に接続された Scheduler Timer はリストが出力されてから 5 秒間待機する（Algorithm 2）．以上の動作により，同じ IP アドレスの URL に対するダウンロードは最低でも 5 秒空くことが保証される．なお，Crawl-delay が設定されている可能性を考え，空のリストを飛ばす際にも 5 秒待機する必要がある（Algorithm 1 の 36 行目）．現実的には Crawl-delay が設定されていない URL でリストが連続して埋まるので，この行が実行されるのはごく稀である．

以上の方法は，Scheduler Timer で待機している間ダウンロードがまったく実行されないという欠点がある．また，1 つのリスト内のダウンロードがすべて完了しなければ次のリストに進まないことから，ダウンロード速度が遅いページがボトルネックになるという欠点がある．しかし，この問題は並列に動作する Scheduler のインスタンス数を増やすことで軽減される．なぜならば，あるインスタンスにおいて処理を待機していても，他のインスタンスではダウンロードが進行中の可能性が高いからである．

また，本スケジューラはフォーカスドクローリングや再クローリングに対応しないが，文献 [8] と同様に PageRank の総和で比較すると，同じダウンロードページ数であれば幅優先よりも効率良く収集できる．ただし RankMass [8] は，PageRank の見積りを計算しながらクローリングしていくため，本手法より良い性能が達成できる．しかし，グラフ構造へのアクセスが必要なため，本手法より計算量で劣る．

本手法の時間計算量は O(1) であり，グラフデータベースへのアクセスも必要ない．IP アドレスのハッシュ値で分割することで容易に並列分散実行することができる．L と H を変化させることでクローリング網羅率と PageRank 優先度のトレードオフを調整できる [20]．

5.2.2　重複 URL の検出

重複 URL の検出は (h) Duplicated URL Checker が持つ LRU キャッシュと BloomFilter で実現する．クローリング中の URL 重複判定は様々なキャッシュアルゴリズムで効率良く検出できることが知られており [3]，本クローラでは LRU キャッシュを用いる．LRU キャッシュからあふれた URL は Bloom Filter [4] に格納して重複削除を行う．Bloom Filter は False Positive を許容する代わりに計算空間量を削減したデータ構造であるため，クローリングされないページが発生する可能性はあるが，クローリング済みであれば必ず重複除去できる．なお，このモジュールは文献 [18] で提案した手法のうち Stable Bloom Filter [5] の近似実装を用いる代わりに通常の Bloom Filter を用いたものである．Stable Bloom Filter は False Positive と False Negative の両方が発生するが，両者の発生率が安定することを保証している．一方 Bloom Filter を用いた場合は False Negative は発生せず確実に重複削除されるが，False Positive の割合は Stable Bloom Filter より悪化する．

アルゴリズムの概要を Algorithm 3 に示した．LRU キャッシュと Bloom Filter を順に確認し，どちらかに含まれていた場合はクローリングせず，どちらにも含まれていなかった場合はクローリングする．LRU キャッシュから追い出された URL は Bloom Filter に追加され必ず重複削除されることが保証される．

5.2.3　ホスト名ごとの情報の保持

名前解決によるオーバヘッドや Web ページへのアクセスごとに robots.txt を確認することを避けるため，ホスト名に対応する IP アドレスや robots.txt の有無といった情報を記録するデータ構造が必要である．本クローラでは文字列に対するトライ型データ構造である Hat-trie [11] を Key-Value ストレージ用に我々で再実装し，(i) Host Data Cache で利用している．

ここでは，DNS キャッシュとしての利用を想定して，ホストを Key とし，1 つの Key ごとに 4 バイトの int 値を Value として格納した場合の性能を示す．実験には公開されている URL データ[*4]に存在するホスト名 5.9 億個，計約 16 GB を利用した．Hat-trie が空の状態から順に Key-Value データを格納していった場合の，メモリ使用量と put 性能の変化を測定した．また，すべての put が完了した後に，ホスト名を順に get する試行を行い，その場合の get 速度の変化を測定した．**図 6** に実験結果を示した．投入データ量と int 値を合わせた量は約 18 GB であるから，ほぼ投入データ量に等しいメモリ使用量になっていることが分かる．URL の共通接頭辞が共有されることで使用するメモリ量を減らすことができ，Java 標準の HashMap のメモリ使用量に対して約 5 分の 1 となった．またアクセス速度も高速であり，1 スレッドでのピーク性能は put は秒間 30 万回，get は 100 万回を超えた[*5]．

Algorithm 3 (h) Duplicated URL Checker

```
1: procedure DuplicatedURLChecker(data: CrawlingData)
2:     if data.url が LRU リストに含まれている then
3:         LRU リスト中の data.url を先頭に移動
4:         return null                          ▷ クローリングしない
5:     end if
6:     LRU リストの先頭に data.url を追加
7:     if LRU リストが容量オーバ then
8:         LRU リストの末尾を削除して BloomFilter に追加
9:     end if
10:    if Bloom Filter に data.url が存在している then
11:        return null                          ▷ クローリングしない
12:    else
13:        return data                          ▷ クローリングする
14:    end if
15: end procedure
```

Host Data Cache モジュールは 4 種類の入力キュー q_0^i, q_1^i, q_2^i, q_3^i を持つ．q_0^i には Duplicated URL Checker からクローリング中に新しく発見された URL を持つ CrawlingData が入力される．ホスト名を用いて Hat-trie を検索し，IP アドレスがキャッシュされていた場合は CrawlingData の ipAddress に格納し，格納されていなかった場合は ipAddress を null にセットする．また robots.txt の有無がキャッシュされているか調べ，有無に従って robotsFlag をセットする．後段の Switcher が ipAddress や robotsFlag の値に基づいて，名前が未解決の場合は Domain Name Resolver，robots.txt を保持している場合は robots.txt Processor，どちらでもない場合は，Scheduler モジュールに転送する．

q_1^i には Domain Name Resolver によって名前解決された CrawlingData が入力される．同じホスト名が現れた場合に備えて ipAddress の値を Hat-trie に書き込み CrawlingData をそのまま出力する．出力は Scheduler の q_0^a に入力される．

q_2^i には Scheduler Timer から出力され，ダウンロードが可

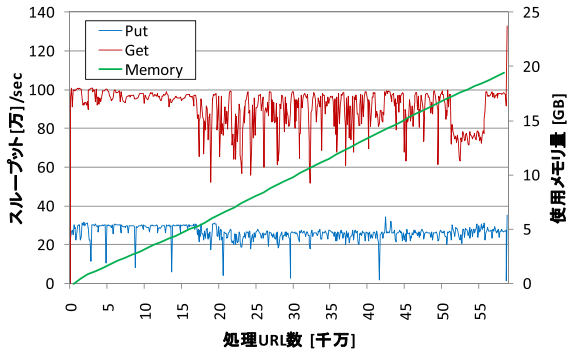


図 6　Hat-trie によるホスト名の格納・取得性能

Fig. 6　The performance of hat-trie for storing and searching host names.

[*4] “Laboratory for Web Algorithmics,” 入手先 ⟨http://law.di.unimi.it/index.php⟩

[*5] 図に見られる性能低下は Java の GC による動作停止であり，停止時間の短い Concurrent GC を利用することで軽減できる．

能になったCrawlingDataが入力される．ここでrobots.txtの有無を再度チェックする．このチェックが必要になるのは，Schedulerモジュール内でクローリングを待機している間に，先行してダウンロードされたURLによってrobots.txtがダウンロードされている可能性があるからである．たとえばScheduler中のリストl_cをダウンロード中に初めて出現した，ある同一ホストのURLが，2つ同時にSchedulerのリストに追加される場合を考えると，Crawl-delayが指定されていなければl_{c+1}とl_{c+2}に追加されることになるが，l_{c+1}がダウンロードされた段階でrobots.txtの存在がチェックされ，l_{c+2}がダウンロードされる段階ではrobots.txtの有無が確定しているからである．

q_3^iにはrobots.txtが見つかったURLに対応するCrawlingDataが入力される．Hat-trieにrobots.txtの存在を記録し，再度スケジューラに送信してページ本体のダウンロードを促す．

5.2.4　robots.txtの処理

robots.txtを処理するモジュールは(k) robots.txt Processorであり，2つの入力キューq_0^kとq_1^kを持つ．Javaの Patternクラスを用いて正規表現に対応した処理を行う．robots.txtではクローラごとにアクセス制限を行うことができるが，安全のために本クローラでは，Disallowが設定されているページはすべてクローリングしないこととした．

q_0^kにはチェック対象のURLが入力され，URLがDisallowに設定されていないか確認を行う．クローリングが許可されていない場合は，CrawlingDataのdownloadableにfalseを設定する．許可されている場合はtrueに設定し出力する．図4の下部中央では，後段のSwitcherがdownloadableフラグを確認し，ダウンロード不可能な場合はSchedulerに終了通知をし，可能な場合はダウンロードを進める．q_1^kには新しくダウンロードしたrobots.txtが入力され，Patternインスタンスを生成して保持する．

5.2.5　ダウンロード処理

ダウンロードを担当するのは(c) robots.txt Downloaderと(d) Downloaderであり，CrawlingDataを入力として受け取り，対象のデータをダウンロードしたあと，CrawlingDataのdownloadedDataにダウンロードしたデータを格納して出力する．ダウンロードにはApache HttpClient[*6]を利用する．robots.txt専用のダウンローダと，robots.txtを除くURL用のダウンローダの2種類に分かれているが，モジュール実装は同一である．

エラーが発生した場合は，CrawlingDataのexceptionに例外が保存され，接続されたSwitcherによってSchedulerのq_1^aに転送され破棄される．本論文の実験ではエラーが発生した場合には再クローリングを行わない．

5.2.6　リンク先の抽出とフィルタリング

HTMLからリンク先を抽出し，クローリングできないURLを判定してフィルタを行うのが(e) HTML Parserと(f) URL Format Filter，そして(g) Explicit URL Filterである．

HTML Parserはダウンロードした HTMLファイルをjsoup[*7]を用いてパースし，リンク先URLを抽出する．Web中に存在するURLは不正確なものも多いため，URL Format Filterは，世界中のドメインリストを集めたPublic Suffix List[*8]に従わないものを不適切なURLとして破棄する．次に，Explicit URL Filterは明示的にクロールしないよう事前設定したURLを除外する．明示的にクローリングしないサイトには，大学が契約しているACM Digital LibraryやIEEE Xplorerなどの電子ジャーナルのサイト，またクローリング中に苦情があったサイトなどを指定することになる．

5.2.7　ダウンロードしたデータの保存

ダウンロードしたデータの保存を担うのが(l) Data Storeである．このモジュールは，受け取ったCrawlingDataのdownloadedDataを，到着順にBSONフォーマットで1つのファイルに格納していく．1つのファイルにまとめるのは，URLごとにファイルを分けるとファイルシステム上に大量のファイルが生成され，データ解析時に高速な読み込みができないためである．後から解析するユーザはファイルを専用ツールで読み込むことができる．ファイルの破損に対応するために，2 GBごとに新しいファイルを作成するようになっている．

5.2.8　URLシード

Webクローラにはクローリング起点となるシードが必要である．クローラの起動時にシードを投入するのが(m) Seederである．Seederは起動時にただ1度だけ動作し，事前に決められたシードURLセットを投入する．本論文ではシードセットの選択方法については論じない．

6.　評価実験

本章では開発したWebクローラの評価実験について述べる．評価では早稲田大学と国立情報学研究所に設置した計算機を用いた．国立情報学研究所に設置された計算機に仮想Web空間を構築し，早稲田大学に設置した計算機から仮想Web空間に対してアクセスするクローリング実験を行った．Webクローラと QueueLinkerはともにJavaで実装されており，コード量はクローラ部分が約6,000行，QueueLinker部分が約13,000行である．実験にはJava 1.7を用いた．

提案Webクローラを動作させる場合，各モジュールにつ

[*6] “HttpComponents HttpClient Overview,” 入手先 ⟨http://hc.apache.org/httpcomponents-client-ga/index.html⟩

[*7] “jsoup Java HTML Parser, with best of DOM, CSS, and jquery,” 入手先 ⟨http://jsoup.org/⟩

[*8] “Public Suffix List,” 入手先 ⟨http://publicsuffix.org/⟩

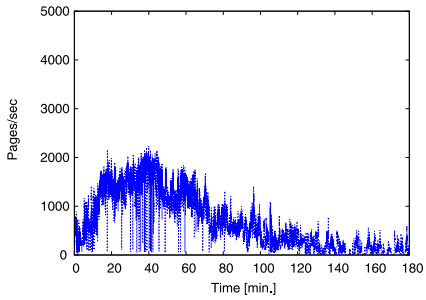


図 7　毎秒ダウンロードページ数（2 台）
Fig. 7　The number of downloaded pages per second (2 computers).

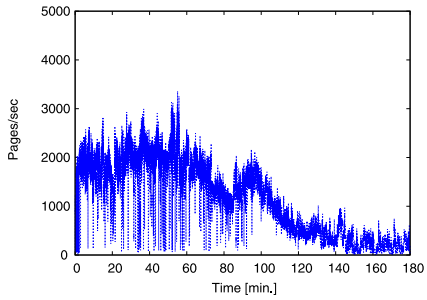


図 8　毎秒ダウンロードページ数（4 台）
Fig. 8　The number of downloaded pages per second (4 computers).

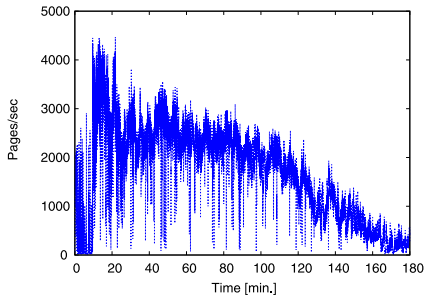


図 9　毎秒ダウンロードページ数（8 台）
Fig. 9　The number of downloaded pages per second (8 computers).

いて，何台の計算機を用いるか，また何スレッドで動作させるか決定する必要がある．最適な配置を求めることは研究課題になると考えられるが，本論文の対象を超えるため扱わない．以下の評価実験では人手で配置を固定して実験を行っている．本実験では Web クローラの理想性能，すなわち最大のスループットを評価するために，Scheduler Timer の待機時間を 0 秒に設定している．

6.1　実験環境とモジュールの配置

実験環境を**図 10** と**表 4** に示す．早稲田大学内にはマスタノード 1 台とクローリングノード 8 台が設置されている．また，国立情報学研究所に設置された計算機 4 台に仮想 Web サーバを構築した．仮想 Web サーバは Web リンクデータの uk-2007-05 [*9] を用い，Web ページを自動生成して仮想の Web 空間を再現するものである．仮想サーバの構築には，軽量かつマルチスレッドで動作できる組み込み向け Web サーバの microhttpd [*10] を用いた．早稲田大学と NII 間は SINET4 [*11] で接続されており，上記計算機が接続されているスイッチのアップリンクはすべて，SINET4 まで 10 Gbps で接続されている．

実験では，Scheduler モジュールを最もメモリがあるマスタノードのみに割り当て，ダウンロードしたデータを保存する Data Store モジュールをストレージノードのみに割り当てた．他のモジュールはすべてのクローリングノードで分散実行するよう割り当てた．実験では 1 つのクローリングノードごとに Downloader を 200 スレッドで並列分散実行し，残りのモジュールは各クローリングノードごとに 1 スレッドで分散実行した．計算機間の通信を削減するために，Downloader がダウンロードした Web ページは同一の計算機上の HTML Parser が処理するようにした．

以上の配置により，ダウンロード待ち URL を多く保持することになる Scheduler モジュールはマスタノードの 512 GB のメモリ空間を利用でき，同様に空間計算量の大きい Duplicated URL Checker や Host Data Cache，robots.txt Processor といったモジュールは 1 台 16 GB のメモリを持つクローリングノードを 8 台利用することで，合計 128 GB のメモリを分散利用することができる．このように，計算機性能がヘテロな環境において，自由なモジュール配置をとることができるのも，本クローラの利点である．

6.2　実験結果

図 7，**図 8**，**図 9** はクローリング開始後 3 時間の 1 秒あたりのダウンロードページ数を示している．計算機台数は 2 台，4 台，8 台で変化させている．計算機台数を増やすに従って収集速度が上昇しており，本クローラの分散処理による性能向上効果を確認できた．

図 11 は，8 台構成ですべてのページをダウンロード完了したあとに，クローリングノードが担当したサイト数と URL 数を示したものである．提案 Web クローラは，機能単位のモジュールで構成されており，それぞれのモジュールをハッシュ分割を用いて並列分散実行するため，担当サイト数と URL 数が計算機間で偏らない．担当数が最も少ない計算機と最も多い計算機での差は，URL が 0.30%，サイト名が 3.0%であった．

一方，**図 12** は比較対象として，Web サイト単位で計算機に処理を割り当てた場合であり，サイトベースの Web クローラに相当する．サイト数の偏りは図 11 と同じになる．しかし，Web サイトが保持する URL 数は偏りがあるため，担当 URL 数は図のように偏る．URL の担当数が最も少ない計算機と最も多い計算機での差は 12.4%となった．

Web サイトそれぞれが保持する URL 数が事前に分かっていれば，各計算機に割り当てる Web サイト数を調整することで，担当 URL 数が均等になるように調整することができる．しかし，Web サイトごとの URL 数はクローリングしなければ分からないため，事前の調整は難しい．この点，提案クローラはハッシュ値によって各処理の分割が

[*9] “Laboratory for Web Algorithmics,” 入手先 ⟨http://law.di.unimi.it/index.php⟩
[*10] “GNU libmicrohttpd,” 入手先 ⟨http://www.gnu.org/software/libmicrohttpd/⟩
[*11] 「学術情報ネットワーク（SINET4，サイネット・フォー）」，入手先 ⟨http://www.sinet.ad.jp/⟩

表 4　各計算機の性能
Table 4　Experimental environment.

	マスタノード	ストレージノード	クローリングノード	仮想 Web サーバノード
CPU	Intel Xeon L7555 ×4	AMD Opteron 8381 HE ×4	Intel Xeon E5530 ×2	Intel Xeon E5530 ×2
メモリ	512 GB	256 GB	16 GB	16 GB
HDD	24 TB	100 TB	2 TB	2 TB
OS	CentOS 5	CentOS 5	CentOS 5	CentOS 5
Network	1 Gbps	10 Gbps	1 Gbps	10 Gbps

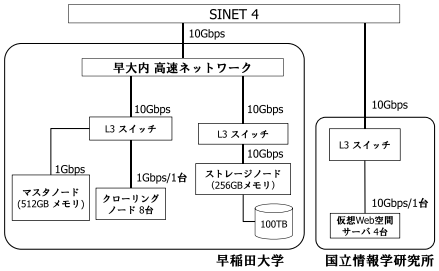


図 10　計算機のネットワーク接続関係
Fig. 10　Network connections between the computers.

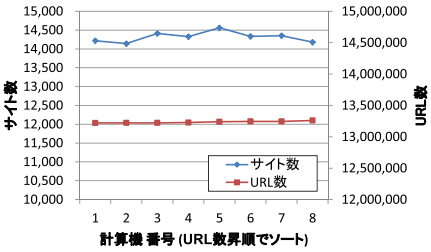


図 11　提案 Web クローラの場合の担当サイト数・URL 数（8 台時）
Fig. 11　The number of handled sites and URLs (our crawler, 8 computers).

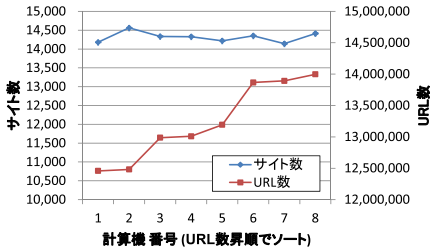


図 12　サイト分割型 Web クローラの場合の担当サイト数・URL 数（8 台時）
Fig. 12　The number of handled sites and URLs (site-based crawler, 8 computers).

可能なため，クローリングを行いながらの負荷分散が実現可能である．本実験は実リンクデータを用いているため，実際のクローリングでの負荷の偏りを評価できていると考えられるが，近年では Twitter や多くの動的ページを持つサイトが増えており，Web サイトごとの URL 数の偏りも変化していると考えられる．より大規模な Web 空間において評価した場合の計算負荷の偏りや，モジュールの最適配置方法は今後検討すべき課題といえる．

7.　おわりに

本論文では我々が開発してきた並列分散 Web クローラについて述べた．本クローラはクローラの機能が Producer-Consumer 型のモジュールに分割されており，我々が開発中の並列分散処理フレームワークである QueueLinker 上で動作する．今後の課題として実 Web 空間でのクローリング評価に加え，Ajax への対応，更新クローリングのスケジューリング，スパムページの判定など，様々な機能の追加があげられる．しかしそれらの機能も，このクローラに対するモジュールとして実装することで実現できると考えている．

なお，本 Web クローラは並列分散処理フレームワークの QueueLinker とともに，今年度中のオープンソース化を目指している．コンパクトでかつカスタマイズ可能なクローラをオープンソース化することで，研究用途はもちろん，様々な目的での大規模 Web データの利用が広がることを期待している．

謝辞　本研究の一部は，文部科学省「Web 社会分析基盤ソフトウェアの研究開発」および科学研究費（挑戦的萌芽研究 No.23650053）によるものである．

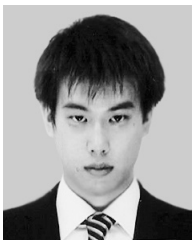
参考文献

[1] Heydon, A. and Najork, M.: Mercator: A Scalable, Extensible Web Crawler, *World Wide Web*, Vol.2, No.4, pp.219–229 (1999).
[2] Koehl, A. and Wang, H.: Surviving a Search Engine Overload, *Proc. WWW 2012* (2012).
[3] Broder, A.Z., Najork, M. and Wiener, J.L.: Efficient URL Caching for World Wide Web Crawling, *Proc. WWW 2003* (2003).
[4] Bloom, B.H.: Space/Time Trade-offs in Hash Coding with Allowable Errors, *Comm. ACM* (*CACM*), Vol.13, No.7 (July 1970).
[5] Deng, F. and Rafiei, D.: Approximately Detecting Duplicates for Streaming Data using Stable Bloom Filters, *Proc. SIGMOD 2006* (2006).
[6] Lee, H., Leonard, D., Wang, X. and Loguinov, D.: IRLbot: Scaling to 6 Billion Pages and Beyond, *ACM Trans. Web* (*TWEB*), Vol.3, No.3, Article 8 (June 2009).
[7] Cho, J., Garcia-Molina, H., Haveliwala, T., Lam, W., Paepcke, A., Raghavan, S. and Wesley, G.: Stanford WebBase Components and Applications, *ACM Trans. Internet Technology* (*TOIT*), Vol.6, No.2, pp.153–186 (May 2006).
[8] Cho, J. and Schonfeld, U.: RankMass Crawler: A Crawler with High Personalized PageRank Coverage Guarantee, *Proc. VLDB 2007* (2007).
[9] Hsieh, J.M., Gribble, S.D. and Levy, H.M.: The Architecture and Implementation of an Extensible Web Crawler, *Proc. NSDI 2010* (2010).
[10] Najork, M. and Heydon, A.: High-Performance Web Crawling, COMPAQ Systems Research Center (Sep. 2001).
[11] Askitis, N. and Sinha, R.: HAT-trie: A Cache-conscious Trie-based Data Structure for Strings, *Proc. ACSC 2007*

(2007).
[12] Boldi, P., Codenotti, B., Santini, M. and Vigna, S.: UbiCrawler: A Scalable Fully Distributed Web Crawler, *Proc. AusWeb 2002* (2002).
[13] Kolay, S., D'Alberto, P., Dasdan, A. and Bhattacharjee, A.: A Larger Scale Study of Robots.txt, *Proc. WWW 2008* (2008).
[14] Shkapenyuk, V. and Suel, T.: Design and Implementation of a High-Performance Distributed Web Crawler, *Proc. ICDE 2002* (2002).
[15] Sun, Y., Zhuang, Z. and Giles, C.L.: A Large-Scale Study of Robots.txt, *Proc. WWW 2007* (2007).
[16] 上田高徳，片瀬弘晶，森本浩介，打田研二，油井　誠，山名早人：QueueLinker：パイプライン型アプリケーションのための分散処理フレームワーク，第 2 回データ工学と情報マネジメントに関するフォーラム（DEIM）(Feb. 2010).
[17] 打田研二，上田高徳，山名早人：カスタマイズ性とリアルタイムなデータ提供を考慮したクローラの設計と実装，第 4 回データ工学と情報マネジメントに関するフォーラム（DEIM）(Mar. 2012).
[18] 久保田展行，上田高徳，山名早人：ウェブクローラ向けの効率的な重複 URL 検出手法，日本データベース学会論文誌，Vol.8, No.1, pp.83–88 (2009).
[19] 村岡洋一，山名早人，松井くにお，橋本三奈子，赤羽匡子，萩原純一：100 億規模の Web ページ収集・分析への挑戦，情報処理，Vol.49, No.11, pp.1277–1283 (2008).
[20] 森本浩介，上田高徳，打田研二，山名早人：ウェブサーバへの最短訪問間隔を保証する時間計算量が O(1) のウェブクローリングスケジューラ，第 3 回データ工学と情報マネジメントに関するフォーラム（DEIM）(Feb. 2011).

上田 高徳　（正会員）

早稲田大学 IT 研究機構研究助手．同大学大学院基幹理工学研究科博士後期課程（在学中）．DSMS，DBMS，並列分散処理，ストレージシステムに関する研究に従事．IEEE，ACM，IEICE，DBSJ 各会員．



佐藤 亘

早稲田大学大学院基幹理工学研究科情報理工学専攻修士 2 年．検索エンジン，並列分散処理に関する研究に従事．DBSJ 学生会員．

鈴木 大地

早稲田大学大学院基幹理工学研究科情報理工学専攻修士 1 年．DBMS，並列分散処理に関する研究に従事．DBSJ 学生会員．

打田 研二

2012 年早稲田大学大学院基幹理工学研究科情報理工学専攻修士課程修了．並列分散処理に関する研究に興味を持つ．

森本 浩介

2011 年早稲田大学大学院基幹理工学研究科情報理工学専攻修士課程修了．並列分散処理・画像処理に関する研究に興味を持つ．

秋岡 明香　（正会員）

2004 年博士（情報科学，早稲田大学）．2004～2005 年国立情報学研究所プロジェクト研究員．2005～2007 年ペンシルバニア州立大学ポスドク研究員．2007～2010 年電気通信大学大学院助教．2010～早稲田大学 IT 研究機構主任研究員．IEEE，ACM，IEICE 各会員．

山名 早人　（正会員）

1993 年早稲田大学大学院理工学研究科博士後期課程修了．博士（工学）．1993～2000 年電子技術総合研究所．2000 年早稲田大学理工学部助教授．2005 年同大学理工学術院教授，NII 客員教授．IEEE, ACM, AAAI, IEICE, DBSJ 各会員．

（担当編集委員　大野 成義）