2011 年度修士論文

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

提出日： 2012 年 1 月 31 日

指導： 山名早人教授

早稲田大学基幹理工学研究科情報理工学専攻

学籍番号：5110B023-1

打田研二

概　要

一般に全ての Web ページを収集することは不可能であり，Web クローラの開発者は目的に応じた Web ページをより早く，より多く集めるためカスタマイズを施している．さらに近年は Twitter や Facebook に代表される情報がより注目を集めており，リアルタイム性の極めて高い情報に対するニーズが増加している．そこで本論文では，以下に示す 2 つの条件を兼ね備えたクローラを提案する．(1).開発者は任意のモジュールの変更・追加・削除が可能 (2).クローラの任意のモジュール間データを，リアルタイムストリームとして受信可能. 従来の多くの Web クローラは，1 スレッドが複数のモジュールの役割を担う設計であり，ソフトウェアアーキテキチャの観点から密結合な設計であった．また，収集した Web データをリアルタイムで提供する例を示した設計は少ない. 本論文で提案する Web クローラは，必要なモジュールを機能ごとに完全に分離させ一切の依存を持たせない．また，全てのモジュールのタスクはモジュール間を通信するメッセージと FIFO キューによって非同期にパイプライン処理を行うことに特徴があり，任意のモジュール間を流れるメッセージをリアルタイムで提供可能である. 本研究では URL 重複除去モジュールのストラテジーの変更と，単語によるフィルタリングを行う新たなモジュールを追加する実験を行うことで，提案する Web クローラがソフトウェアアーキテクチャの観点から租な設計となっていることを確認した．また，任意のモジュール間データをリアルタイムストリームとして活用できることを確認するため，DNS 対応表を作成するアプリケーション，指定された単語をカウントするアプリケーションへモジュール間データを流れるストリームをリアルタイムで提供できることを確認した.

目　次

第1章　はじめに

昨今，Web 上のデータが活用される場面が増加している．画像に関する研究[1]や言語に関する研究で活用される[2]こともあれば Web 上のデータからキャンペーン効果の測定や市場調査を行うサービス[3]も存在している．また，過去の Web ページの閲覧が可能なサービスも存在しており[4]，Web 上のデータに対する需要は高まる一方である．このような研究やサービス，ビジネスを行うにあたり，Web 上に存在するデータを収集するプログラムである Web クローラは必要不可欠な存在となっている. それを受けて Apache Project の一つである Lucene[5]とそのサブプロジェクトである Nutch[6]，Solr[7]， YaCy[8]のようなオープンソースの Web クローラ・検索エンジンの開発が盛んであり，Web クローリングそのものを提供するサービス[9]も存在している．

また，新たにインターネットを利用したサービスとして Social Networking Service(以下 SNS)が発達し，より多くのユーザに受け入れられている．SNS の 1 つである Twitter はユーザ層の幅広さだけでなく，その情報のリアルタイム性の高さが特に注目を集めている．例えば 2011 年 3 月 11 日の東日本大震災時には，日本の各地域から多くの最新情報が Twitter 上で発信され，多くの Twitter ユーザが自分の望む情報を入手することができた．

可能であれば，Web 全体が持つ莫大な情報と，Twitter に代表されるリアルタイム性を兼ね備えた情報を，柔軟に収集し，活用できることが望ましい．ここでの柔軟さとは，収集する Web ページの内容をフィルタリングする[10][11][12]，あるいは収集する Web ページの優先順位を変更する[13][14]といった Web クローラのカスタマイズやストラテジーの変更が容易であることを意味する．一方で Web クローラが収集した情報を利用するエンドユーザの視点に立った場合，Twitter から提供される Streaming API[14]のように，より必要な情報だけを，よりリアルタイムに受け取り活用できると望ましいケースも想定される．ここでのエンドユーザとは，単純に検索エンジンを利用する様な一般的なユーザだけでなく，Web 上の情報を監視・調査する研究者や Web アプリケーションを作成するエンジニアといった幅広い層を想定している．

しかし，Nutch や Solr のオープンソースクローラはモジュール間の依存が強く，ストラテジーの変更や新たなモジュールを追加することが難しい設計となっている．また，80legs によるサービス等は，集めたデータはファイルに纏められた後にユーザに提供され，Streaming API 程のリアルタイム性は期待できない. 2010 年に発表された Jonathan[16]らの設計では，ユーザは任意の文字列によるフィルタ(正規表現も可能)をクローラに登録することでフィルタにマッチする Web ページのテキストをリアルタイムにエンドユーザに提供できる，しかし，エンドユーザが求める Web 上のデータは，Web ページのテキストデータだけとは限らない.

そこで本論文では，上記に挙げた要求を満たすため，以下に示す 2 つの条件を兼ね備えたクローラの設計を提案する.

1. 開発者は任意のモジュールの変更・追加・削除が可能

2. エンドユーザはクローラの任意のモジュール間データを，リアルタイムにストリームとして受け取ることが可能

我々の提案する設計は，クローラに必要なモジュール間の機能的な依存関係を全て無くし，全てのモジュールのタスクはモジュール間を通信するメッセージと FIFO キューによって非同期に管理することができる．そのため，1 つのモジュールの機能変更は，他のモジュールへ一切の機能面での影響を与えない．またエンドユーザは適切なモジュールにリッスンポートを立てて接続するだけで，自分が望む任意のモジュール間のデータを入手することが可能である.

本論文は以下の構成をとる．まず，第 2 章で Web クローラについて説明し，第 3 章で関連研究について説明する．第 4 章で提案するパイプライン型 Web クローラの設計について説明し，第 5 章で Web クローラの具体的な構成と実装について述べる．第 6 章で実験と結果について述べ，最後に第 7 章にてまとめる.

第2章　Web クローラ

Web クローラ(別名 Web Spider)は Web 上を巡回し，世界中に存在する Web 上のデータを自動的に収集するプログラムを指す．本章では Web クローラに必要とされる条件，Web クローラを構成する要素についてまとめる．

2. 1 Web クローラに求められる条件

クローラの 1 つの最大の目標は，ある瞬間に存在する Web 上のデータを全てダウンロードして記録することである．実際にはネットワーク帯域，CPU，Memory，ディスク I/O といったハードウェア面の制限に加え，動的に生成される Web ページも存在するため，Web 空間全体のスナップショットを捉え保存することは不可能である．現実には，

1. 可能な限り多くの Web ページを収集する
2. より高速に Web ページを収集する
3. 相手 Web サーバへの負荷を考慮する
4. robots.txt[17]に従ってデータを収集する．

といった条件が Web クローラの最低限必要な項目として挙げられる．例えば検索エンジンはユーザに幅広い情報を提供する必要があるため，可能な限り Web データをより多く，より早く収集する必要がある．また，研究や調査目的で Web データを使用する際にも，一般的にはより多くのデータを用いる方がよい．しかし，高速なクローリングを行う際には，クローリング対象の Web サーバに負荷を掛けすぎないようにすること(politeness policy)も重要である．通常，politeness policy はどの Web ページにいつアクセスを試みるかをスケジューリングすることで実現される．また，各 Web サーバのドメイン直下に robots.txt が設定されている場合は，Robots Execution Standard(以下 RES)プロトコルに準じてクローリングを行

うことが推奨されている．RES は正式には拘束力を持たないとされているが，商用検索エンジンを代表する Googlebot， Bingbot/MSNbot，Yetibot，Baiduspider といった全ての Web クローラは RES に対応するように努めていると公式サポートページに記載している．また，テキサス大学の Lee らによって作成された IRLBot[18]や，スタンフォード大学で作成された WebBase[19]といった Web クローラはいずれも RES に対応しており，相手 Web サーバに対する配慮に努めている．

さて，実際には 2008 年 7 月時点で，1 兆を超える web ページが世界中には存在していると報告されている[20]．日々変化を続ける莫大な量の Web ページを全て収集することは不可能なので，望む Web データを優先的に収集できるように，様々な Web クローラが考案されてきた．例えば下記に示すトピックに関する研究は特に盛んである．

5. 望むコンテンツの Web ページを優先して収集する[10][11][12]

6. 質の高いとされる Web ページを優先的に収集する[13][14]

7. 収集した Web ページを最新の状態に保つ[24][25][26]

5-7 に関しては，エンドユーザが一体何のために，どのようなデータを利用したいかによって扱うストラテジーが異なってくる．そのため，Web クローラは柔軟にストラテジーのカスタマイズが可能な設計であることが望ましい．

2. 2 Web クローラの構成

Web クローラを構成する要素は様々なものがある．例えば 1998 年に発表された Mercator[21]の構成を参考に，基本的な Web クローラの構成を図 1 に示す．

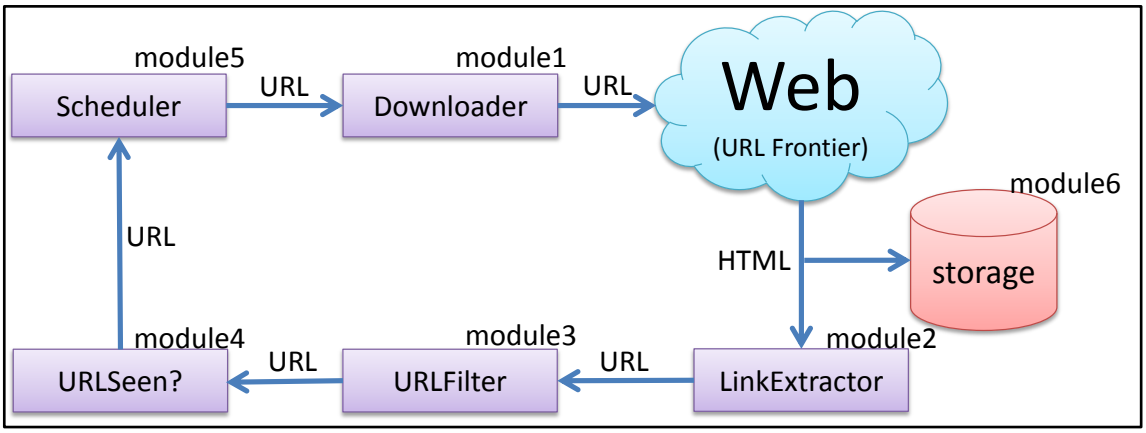

図 1　基本的な WebCrawler の構成例

図 1 に示すように, Web クローラは module1～module6 に示されたプロセスを繰り返し処理するソフトウェアである．バッチ処理のように逐次処理をそれぞれのプロセスで実行するのでなく，ストリーム処理として，それぞれの処理をパイプライン的に非同期に実行することが可能である．図 1 の各 module の要素は以下の処理を担っている．

module1.　URL に対応した Web データをダウンロード

module2.　ダウンロードされたデータから新たな URL を抽出

module3.　対象 URL をダウンロードしてよいかチェック

module4.　既にクロール済みの URL を除去

module5.　次にダウンロードすべき URL の決定

module6.　ダウンロードしたデータの保存

また，より実用性の高いクローラを構成するために，上記の要素以外にも以下の機能を持つ module を組み込んだクローラの構成が考えられる．

module7.　ダウンロードしたデータの重複除去

module8.　スパムページの判定・削除[22][23]

module9.　収集済み URL のデータ更新[24][25][26]

module10.　エンドユーザへのストリームデータの提供[16]

module11.　Hidden Web のような空間もクロールする[27]

module 7 以降の要素は，クローリングの目的によって選択的に変更・追加・削除されるべきである．例えば，スパムページを多く調査したいのであれば，module8 によってスパムページを判定し，該当した Web ページ周辺を集中的にクローリングするとよい．また，最新の Web 上のマルウェアを検出したいのであれば，例えば Clam virus database[28]から公開されているマルウェアのパターンにマッチした Web ページをモジュール 10 と組み合わせ，リアルタイムにマルウェアを解析するエンドユーザ，あるいはアプリケーションにデータを提供すればよい．

2. 3 まとめ

本節では Web クローラに求められる条件と，Web クローラの構成を説明した．2. 1 節と 2. 2 節で示したように，柔軟なモジュールの変更・追加・削除が可能であり，エンドユーザの目的に応じた Web データを提供できる設計がより実用的な Web クローラを構成する上で望ましい．

そこで本研究では，

1. 開発者は任意のモジュールの変更・追加・削除が可能
2. エンドユーザはクローラの任意のモジュール間データを，リアルタイムにストリームとして受信可能

という 2 つの条件を満たす Web クローラの設計を提案する．

第3章　Web クローラ設計の既存研究

本章では，既存研究でどのような Web クローラが設計されてきたのかを分類し，まとめる．

3. 1 概要

2. 1 節で説明した通り，Web クローラは多くの条件を満たす必要がある．また，2. 2 節で説明した通り，クローラは様々な観点からアプローチがなされ，開発されてきた．3. 2 節では初期の Web クローラについて触れ，3. 3 節では高速性とスケーラビリティに重点を置いたクローラの構成について説明する．3. 4 節ではクローリングしたデータをどのようにしてエンドユーザに提供するかについての既存研究について説明し，3. 5 節では再クローリングの設計について説明する．最後に 3. 6 節でまとめる．

3. 2 初期の Web クローラ設計

本節では, 1994 年に提案された最初期の Web クローラの 1 つである The RBSE Spider[29] について説明する．

The RBSE Spider は幅優先探索法によってスケジューリングする方法, 辿る深さを限定してスケジューリングする方法，Web ページを収集した後のインデックス作成方法，データベースとの連携といった現在の Web クローラの基礎となる設計部分が述べられている．

また，この頃の Web クローラの収集ページ数は数万から十数万であり，今日対象となる規模の Web ページ数と大きく異なっている．そのため，概してこの年代，あるいはそれ以前に提案された Web クローラを単純にそのまま現在のクローラへ利用することは実用的ではない.

3. 3 高速性・スケーラビリティを意識したクローラデザイン

Web クローリングの速度やスケーラビリティを向上させるために，これまでに非常に数多くの研究がなされてきた．並列・分散処理に関したアプローチが一般的であるが，IRLBot[18]のように 1 台の計算機で非常に高速にクローリングする研究も為されてきた．本節ではこれらのクローラの主な設計について説明する．

3. 3. 1 Mercator[21]

Mercator とは, 1999 年に Heydon らによって実装された Web クローラである. Mercator は Worker スレッドがそれぞれ非同期に，並列に実行できることに大きな特徴がある．1 つの Worker スレッドは，ダウンロード，同一 Web ページを収集済みでないか，リンク抽出，URL のフィルタリング，URL の重複チェックという働きを持つ複数のコンポーネントのタスクを担う構成となっており, 新たなコンポーネントを追加することも容易としている．しかし，worker スレッドのコンポーネント間で実際にどのようなデータが受け渡されるかに関しては詳しく言及されていない．本提案手法は，1 つのスレッドは 1 つのコンポーネントだけのタスクを担う点と，コンポーネント間のタスクの受け渡しを全てメッセージによって行う点で大きく違っている．

3. 3. 2 An Adaptive Model for Optimizing Performance of an Incremental Web Crawler[30]

Edwards らによって提案された設計は IBM が開発した Web Fountain Crawler の設計に類似している．Web Fountain Crawler は大きく 3 つのコンポーネントに分類できる．

1. ダウンロードするコンポーネント(Ants)
2. DNS 解決するコンポーネント

3. クローリングをコントロールするコンポーネント

1 と 2 のコンポーネントに関しては非同期に分散実行が可能であり，高速且つスケールするように設計されている．またコンポーネント間のデータの受け渡しは MPI(Message Passing Interface)によってストリーム的に行われており，その点で本研究と類似している．しかし，3 のコンポーネントに関しては，その中にスケジューリング，HTML パーシング，URL 重複除去などのコンポーネントが含まれており，3 の内部コンポーネント間のデータの受け渡し方法は不明である．また，3 のコンポーネントについては分散処理ができない設計となっており，本設計よりもコンポーネント間の結びつきが強い内部実装となっている．

3. 3. 3 UbiCrawler[31]

UbiCrawler は，Boldi らによって提案された Peer to Peer による分散クローラである．UbiCrawler の特徴は Consistent Hashing を用いた Peer to Peer による完全分散であり，拡張性と耐障害性に優れている．また，Java で実装されたオープンソースソフトウェアとして UbiCrawler を公開している．

全てのタスクが分散実行可能とされているが，実際に論文中で論じられているのは主にスケジューリングとダウンロードの分散処理が可能な設計となっている部分である．リンク解析や重複削除といったモジュールを追加する必要性が生じた場合の考察，あるいはダウンロードしたデータをどのようにエンドユーザに提供するかに至るまでは言及されていない．

3. 3. 4 Design and Implementation of a High-Performance Distributed Web Crawler[32]

2002 年に Shkapenyuk らによって考案されたクローラで，本論文と非常に近い主旨を持って設計されたクローラである．Shkapenyuk らの設計[32]を図 2 に示す

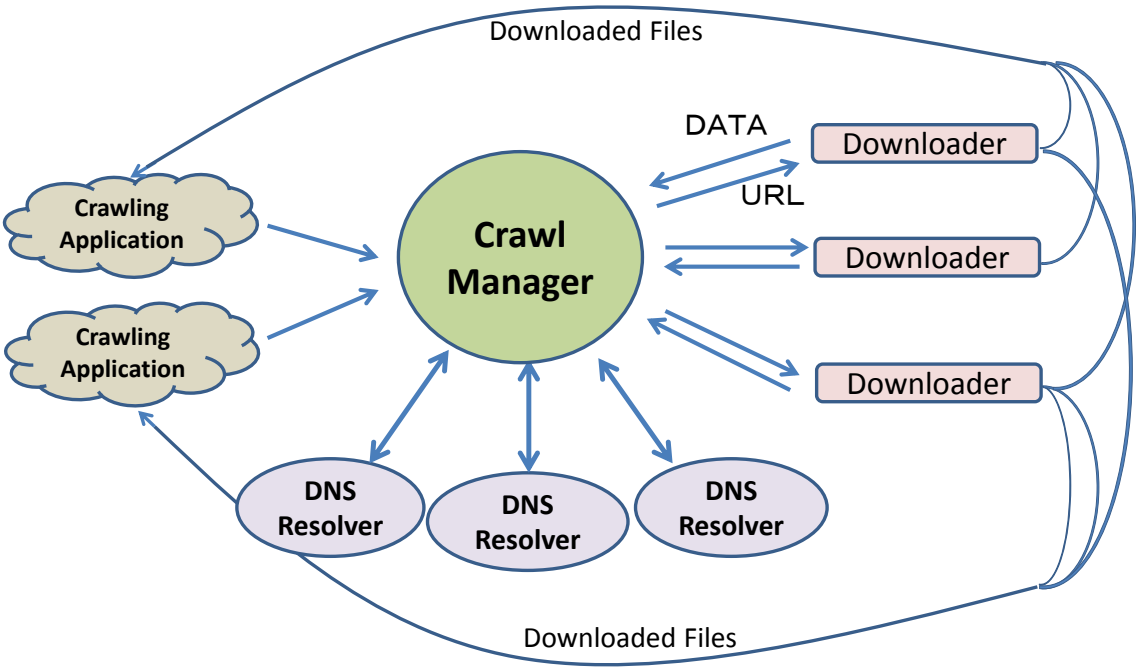

図 2　Shkapenyuk の Web クローラの設計図

([32]Fifure.2:small crawler configuration を参考)

名前解決，Web ページのダウンロードの 2 つのタスクを非同期に実行できる設計であり，そのコンポーネント間のタスク伝達はメッセージによって行われる．また，ダウンロードした Web ページのデータをエンドユーザアプリケーションへストリーム形式で提供できる設計にもなっている．本提案手法と Shkapenyuk らの手法と大きく異なる点は，図 2 の Crawl Manager に該当する部分である．提案手法では Crawl Manager に該当する機能も完全にモジュールが分離されており，メッセージによるタスク伝搬を徹底している．それに伴い，提案手法は任意のモジュール間を流れるデータをエンドユーザにストリーム形式で容易に提供できる．Crawl Manager に備わっている機能としては，どの Web ページをどのタイミングでアクセスするかを決定するスケジューリング機能のモジュール，あるいは URL の重複チェックモジュールといった Web クローラに欠かせない機能等が列挙される．

3. 3. 5　IRLbot[18]

IRLbot は Lee らによって提案された高速化を目指した Web クローラであり，かつ前述の Web クローラと異なり，単一の計算機でクローリングを行っている．しかし単一の計算機

でのクローリングでありながら，毎秒 1,789 ページと 1 台の計算機当たりでは最も高速な収集を行っており，収集規模も 60 億と大規模である．ソフトウェアデザインというよりは，各モジュールの機能説明にフォーカスしている．

3. 4 エンドユーザへのデータ提供を意識したデザイン

Web 上のデータを活用するためには，Web クローラによって収集されたデータをどのようにしてエンドユーザへ提供するかを含め考えなければならない．本節では，データベースに登録する従来の検索エンジン型の手法[19]と，Jonathan らによるエンドユーザへリアルタイムストリームとして提供できる設計を説明する．

3. 4. 1 Stanford WebBase Components and Applications [19]

Cho らは，Web を解析するためのツールとして WebBase を開発した．WebBase に含まれるアプリケーションの一つとして Web クローラが搭載されており，Web クローラのデザイン，収集したデータの提供の仕方両方について考察をしている．WebBase のクローラはサイトクローラとイベントループによるシングルプロセスによる複数サイトに対する Web クローラの組み合わせで構築されている．サイトクローラとは，Web サイトごとにクローリングを実行する Web クローラである．ただし，イベントループによるシングルプロセスの複数サイトに対するクローラは，イベントループが単一障害点になりやすい可能性がある．

また，収集したデータに関しては，分散データベースに蓄えインデックス化を行っており，ユーザのクエリに応じて高速に該当データをレスポンスできる検索エンジン型の設計となっている．WebBase のクローラは，スループットがクローリングプロセス数に対してほぼ線形にスケールしており，最大で 40 プロセス時に毎秒 6000 ページで収集している．また，リポジトリから検索されたクエリに対応するデータをストリームとして流す際には，約 4,000Page/s の速度で送信できると報告している．

3. 4. 2 The Architecture and Implementation of an Extensible Web Crawler[16]

一般ユーザにとっては，クローラによって収集された全てのデータは必要でない．例えば「早稲田大学」についての Web 全体の情報が欲しいのであれば,「早稲田大学」あるいはそれに関連したフレーズを含む Web ページだけをクローラから提供して欲しいと考えるのが一般的である．2010 年に Jonathan[16]らは，エンドユーザが望むデータをフィルタリングし，クローラによって収集されたテキストデータをリアルタイムでストリームデータとしてエンドユーザに提供できるシステムの設計の一例を示し，実現している．

検索エンジンは，Web ページを予めインデックス化しておく．そして，クエリに応じたインデックスを検索し，該当 Web ページをランキングしてレスポンスとして返す．一方 Jonathan らは，ユーザが望む全ての情報を予めインデックス化しておき，Web ページをダウンロードした際にその Web ページをクエリ群のインデックスと参照する．そして一致したクエリを発行したユーザに，該当する Web ページをストリームとして提供する．しかし，Jonathan らが対象としているのは Web ページのデータのみであり，robots.txt の情報や最新の DNS の対応表を作成したいといったユースケースも想定され,その場合は対応できない．これに対し，提案手法では任意のモジュール間のデータをリアルタイムストリーム形式でユーザに提供でき，より柔軟性に優れている．また，Jonathan は画像や動画といったデータ(richer media types)を将来提供できることを課題としている．提案する Web クローラであれば，それらの要求に対応したモジュールを追加することで容易に対応できる．

3. 5 Web ページの再収集を目的としたデザイン

Web ページの再収集は，Web ページをできる限り新しい状態に保つために行われる．主に検索エンジンに組み込まれるソフトウェアとしての Web クローラで行われる．

3. 5. 1　多周期的更新アクセスに適した二次記憶管理技法[25]

田村らの手法は Web ページごとに独立して更新間隔の推定と再アクセスのスケジューリングを行う．また，同一の Web サーバに属する Web ページに逐次にアクセスするために，Web ページごとのアクセス間隔をもとに Web サーバごとへのアクセス間隔を決定する．エンドユーザへはアーカイブとしてデータを提供できることを目標とした設計で，扱うデータがスケールするように設計されている．

3. 6 まとめ

本章では，これまでに Web クローラがどのようなことを考慮し，設計されてきたかをまとめた．表 1 に，3. 3 節～3. 5 節で述べた既存研究と，本研究との比較をまとめる．表 1 中の Message Passing とはコンポーネント間のデータの受け渡しをメッセージ処理行うことを示しており，共有のメモリ領域を用いる必要性がないことを意味している．検索エンジン型とはダウンロードしたデータをデータベースにインデックス化して追加し，そのインデックスを基に情報にアクセスする手法を意味している．

表 1　関連研究まとめ

手法	年度	コンポーネント間データのやりとり	エンドユーザへのデータ提供	備考
[21]	1999	メモリ領域共有	不明記	worker スレッドが複数のコンポーネントの機能を果たす．並列実行可能.
[30]	2001	メモリ領域共有と Message Passing	不明記	MPI によって一部のコンポーネント間がメッセージによってタスク伝搬されている．分散実行時にノード間通信を行う.
[31]	2002	メモリ領域共有と Message Passing	不明記	拡張性高い設計となっている．しかし，全く新しいコンポーネントが追加するケース等は考察されていない.
[32]	2002	メモリ領域共有と Message Passing	不明記	Crawler Application が複数のコンポーネントから成るタスクを行う.
[18]	2008	メモリ領域共有と Message Passing	不明記	1 台のノードで高速にクローリングし，スケールするためのコンポーネントの実装詳細について述べている,
[16]	2010	不明記	リアルタイムストリーム	エンドユーザに，指定されたクエリにマッチしたデータをリアルタイムストリームとして提供できる設計．クエリには正規表現を用いることもできる.
[19]	2006	メモリ領域共有と Message Passing	検索エンジン型	検索されたクエリに対応するデータをリポジトリからストリームとして流す際に，約 4,000Page/s の速度で送信できる.
[25]	2010	メモリ領域共有	アーカイブ	保存する Web データを最新の状態に保つ再クローリングの設計にフォーカスしている.
提案手法	2012	Message Passing	リアルタイムストリーム	1 つのスレッドは 1 つのコンポーネントのタスクを担う．タスクの伝達は全てメッセージ．任意のコンポーネント間のデータをユーザへリアルタイムストリームとして提供可能.

表 1 から分かるように，クローラ全体の設計について論じていても，エンドユーザが Web クローラによって収集されたデータをどのように利用するかまで意識した設計は少ない．また，多くの既存の Web クローラは 1 つのスレッドが複数のモジュールのタスクを担う設計となっており，モジュールの仕様変更，あるいは新しいモジュールの追加，削除をする際に他のモジュールへ与える影響が懸念される．

我々が提案する Web クローラは，全てのモジュールは他のモジュールと依存関係を一切持たない．そのため，

1. 開発者は任意のモジュールの変更・追加・削除が容易に可能

となっている．また，モジュール間のメッセージをエンドユーザに提供することが極めて簡単な設計となっており，

2. エンドユーザはクローラの任意のモジュール間データを，リアルタイムにストリームとして受け取ることが可能

となっている．

第4章　パイプライン型 Web クローラの提案

本章では上記に挙げた項目を満たす Web クローラを作成するため，パイプライン型の処理が可能な Web クローラを提案する．提案する Web クローラは全てのモジュールが完全独立な設計であり，それぞれのタスクを非同期に実行することができる．タスクの伝搬は全てメッセージによって行われる．4. 2 節では検証用に開発した Web クローラの設計について説明する．本実験のために作成した Web クローラの実装に関しては第 5 章で説明する．

4. 1 要求条件

本手法が目指す Web クローラは前述の通り下記の 2 点を満たす Web クローラを作成することを目標とする．

1. 開発者は任意のモジュールの変更・追加・削除が可能
2. エンドユーザはクローラの任意のモジュール間データを，リアルタイムにストリームとして受け取ることが可能

4. 2 提案手法

提案する Web クローラは，マルチスレッドプログラミングのデザインパターンのひとつである Producer/Consumer パターンを基にして構成される．構成する各モジュールを FIFO キューで接続し，非同期にデータやメッセージの受け渡しを行うことでクローリング動作を実現している．

実際に我々が作成した Web クローラのモジュール間の接続関係を図 3 に示す.我々のクローラはスレッド 1 つがモジュール 1 つの役割を果たす．そのため，各モジュールに割り当てるスレッド数を柔軟に変更できるという利点も挙げられる．例えば，Web ページのダウ

ンロードを行うモジュールと，Web ページからリンク先 URL の抽出を行うモジュールのスレッド数を独立して調整することができる．そのため，CPU 負荷に合わせてスレッド数をモジュール単位の粒度で最適化できる．また，FIFO キューを用いることで新たなモジュールの追加や削除，接続順の変更も容易に可能となる．本実験で各モジュールを流れるメッセージの中身は，主に図 4 のメンバーを持つデータである．本実験のモジュールで必要なデータは URL, IP アドレス, Web データ(HTML or IMAGE)なので図 4 に示す ***StreamWebData*** というデータ構造とした．この ***StreamWebData*** のデータ構造はあくまでも一例であり，目的に応じて変更可能である．

各モジュールに割り当てられたスレッドは図 5，図 6 で示すように，自分のタスクとして渡された ***StreamWebData*** から必要な情報を参照して処理をする．そして，処理した結果を新たな ***StreamWebData'***として次の接続先モジュールに伝搬させる方式を取る構成となっている．例えば，Parser であれば ***StreamWebData*** の中から HTML データを抽出し，パーシングを行う．そして，パーシングされた URL を新たな ***StreamWebData'***として接続先のモジュールに流す．また，重複除去モジュールであれば ***StreamWebData*** 中から URL を参照し，既に巡回済みの URL であれば ***StreamWebData*** を破棄し，そうでなければ次のモジュールに ***StreamWebData'***としてそのまま流す．なお，モジュールの処理をマルチスレッドで行う際には，図 6 中の Queue に関わる処理は全て同期するように構成する．

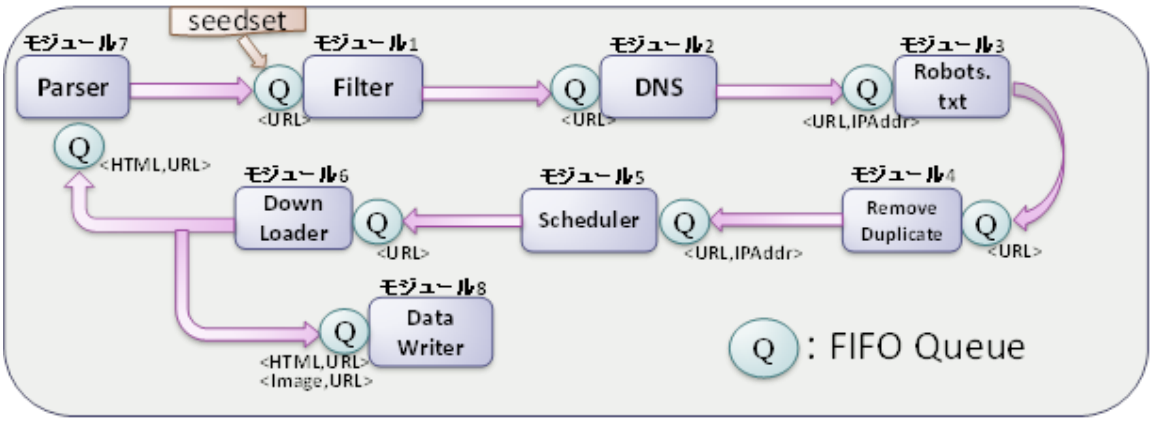

図 3 パイプライン処理により構成される Web クローラ

Data 構造 : StreamWebData

```
struct StreamWebData {
    URL url;            // URL
    byte[] ipaddress; // 対象 url の IP アドレス
    String extension; //何のファイルかを示す拡張子
    byte[] data;        // 対象 URL 先のデータ
}
```

図 4 ストリームで流れるデータ

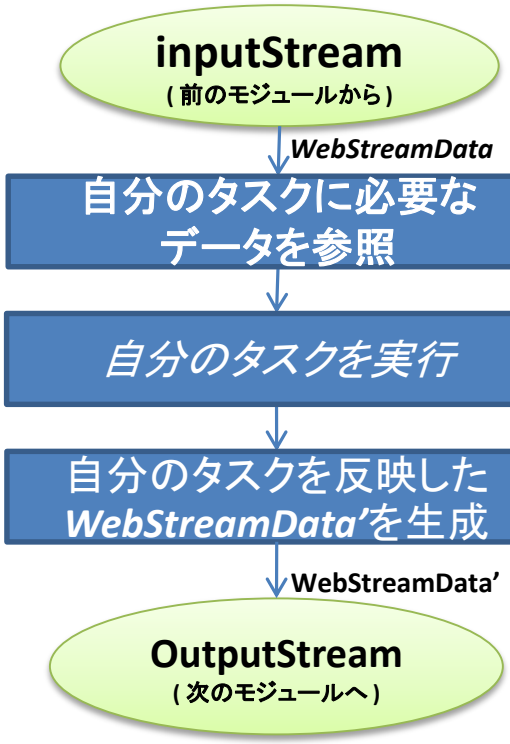

図 5　モジュールのタスク処理フローチャート

Algorithm : 各モジュールの処理

```
Queue<WebStreamData> inputQueue; //  このモジュール内で処理すべきデータ
Queue<WebStreamData> outputQueue; //  次のモジュールで処理すべきデータ
While (true) {
    WebStreamData data = inputQueue.take();
    WebStreamData nextWebStream = executeOwnJob(data); //自分のタスクを実行
    followToNextNode(nextWebStream); //  次のノードへタスクを流す
}
```

図 6 各モジュールの処理

4.3 まとめ

本章ではパイプライン型 Web クローラの設計を説明した. 本手法では, Producer/Consumer パターンを基にして，Web クローラを構成するモジュールを完全に分離している．全てのモジュールのタスクはモジュール間を通信するメッセージと FIFO キューによって非同期に管理することができ，1 つのモジュールの機能変更は，他のモジュールの持つ機能へ一切の影響を与えないことを保証することができ，ソースコードの変更も生じない．モジュールを新たに追加したい際には，開発者は FIFO キューの接続関係だけを変更するだけである．また，エンドユーザは適切なモジュールにリッスンポートを立てて接続するだけで，自分が望む任意のモジュール間のデータを入手することが可能な設計となっている．さらに，FIFO キューでデータを扱っているため，メモリに収まらなくなったデータはファイルに出力する機構を設けることで，流れるデータの肥大化にモジュール単位で対応することができる.

第5章　パイプライン型 Web クローラの実装

本章では，提案する Web クローラの実装に関して記述する．なお，第 6 章の一部の評価実験の際に，本章の 5. 1. 1 節～5. 1. 7 節で説明する一部のモジュールに関して機能変更を加えて評価実験を行っている．実験の際のモジュールの変更に関する詳細は第 6 章にて適宜記述する．

5. 1 構成するモジュール

本クローラを構成するモジュールは，(1) URL フィルタモジュール，(2) 名前解決モジュール，(3) robots.txt 処理モジュール，(4) URL 重複削除モジュール，(5) スケジューラモジュール，(6) ダウンローダモジュール，(7) HTML パーサモジュール，(8)ファイル出力モジュールの 8 種類である．各モジュールは FIFO キューで環状に接続され，パイプライン効果で並列に処理を行うことができる．図 3 に接続関係を示す．図 3 に示す各モジュールのキューの下側にある表示はそのモジュールが自分のタスクを行うのに必要なデータの種類である．各モジュールは自分に必要なデータさえ取得すれば，自分のタスクを実行することができる．そして次のモジュールのキューに対し，自分のタスクの結果に応じたメッセージを流す push 方式をとる．以下，各モジュールの機能について順に概要を述べる．

5. 1. 1　URL フィルタ

本モジュールでは RFC1738[23]に従わない不正な URL の除去と，存在しないトップレベルドメインの URL を除去する．また，ユーザが明示的にクローリングの対象外に指定した URL を検出し除去する．すなわち，モジュール 7 のパーサから流れてきた URL をチェックして，対象外の URL はモジュール 2 へ流さずにシャットダウンし，クローリングする対象を制御する役割を果たす.

5. 1. 2　名前解決

本モジュールは, URL の名前解決, すなわち URL から IP アドレスへの変換を担う. DNS サーバへの過剰な負荷を避けるために，一度名前解決した IP アドレスはローカルにキャッシュする. この DNS キャッシュには, トライ型データ構造である HAT-trie[24]を用いている. HAT-trie は文字列検索用のデータ構造であり，我々は URL から IP アドレスを検索できるよう拡張している．図 7 に DNS モジュール内でのデータの流れを実線の矢印で示し，DNS キャッシュに対する問い合わせを破線の矢印で示した.

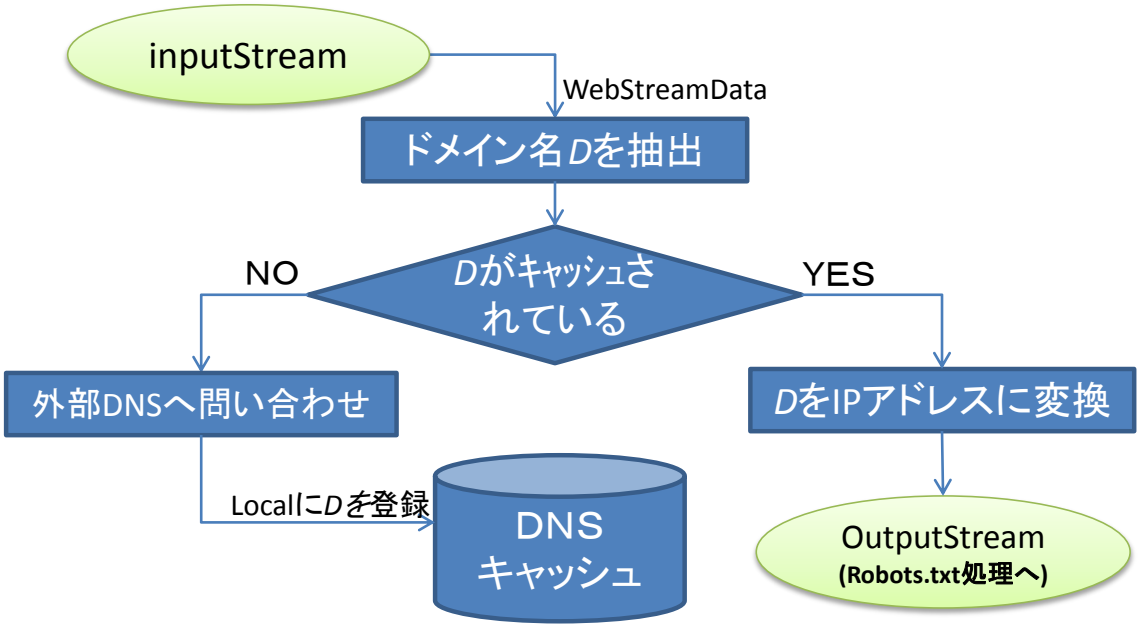

図 7 名前解決の流れ

5. 1. 3　robots.txt 処理

本モジュールは robots.txt[17]の記述を処理してクローリングの制御を行う．Web ページの管理者は，クローラによるアクセスを制限あるいは許可するために，ホストのルートディレクトリに robots.txt を配置することができる．robots.txt には，ホスト内のディレクトリごとにクローリングの許可・禁止を正規表現で指定することができる．図 8 に，robots.txt の処

理の流れを矢印で示し，分岐条件を併記した．対象 URL のクローリングが robots.txt で制限されているなら，その URL を破棄し，許可されているなら次のモジュールへ送信することで管理者の意図に沿ったクローリングができる．robots.txt にサイトマップを記述することにより Web クローラに URL の追加情報を与えることが可能であるが，本 Web クローラの実装ではサイトマップは考慮に入れていない．

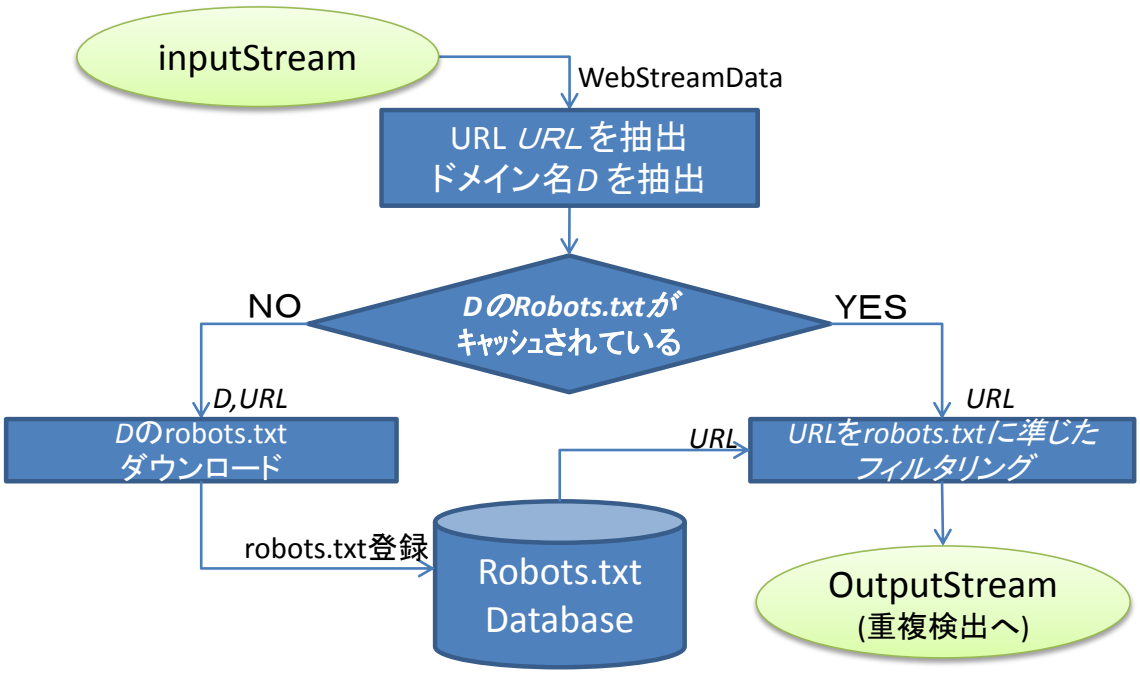

図 8robots.txt の処理の流れ

5. 1. 4　URL 重複検出

ダウンロード済みの URL を再びダウンロードしないように URL の重複削除を行う．図 9 に URL 重複削除の流れを矢印で示し，重複削除の条件を併記した．本クローラでは高速化のため，我々が過去に開発した手法[38]をもとに，LRU キャッシュと Bloom Filter を組み合わせて近似的に重複削除を行う．Bloom Filter とは偽陽性を許容する代わりに O(1)で重複判定を行うことができるアルゴリズムである．ここでの偽陽性とはクロールしていない URL に対して，クロール済みであると判定してしまうことである．詳しくは論文[38]を参照されたい．

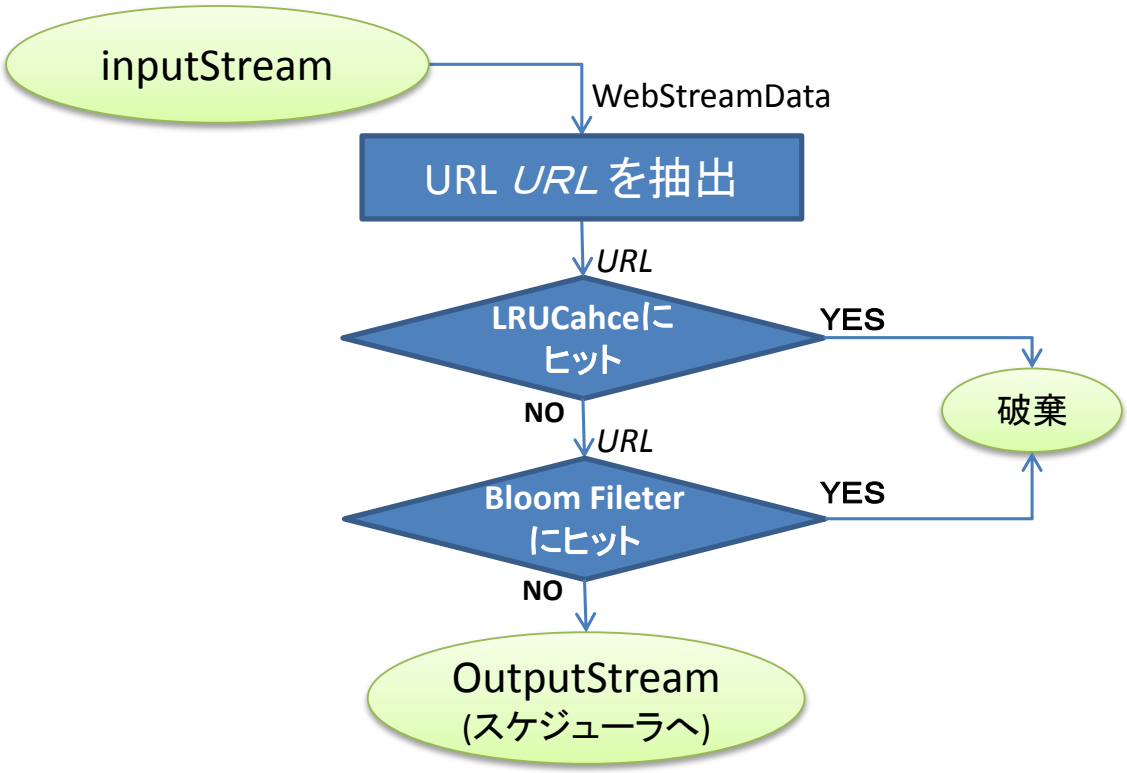

図 9URL 重複削除の流れ

5. 1. 5　スケジューラ

本モジュールは，我々が過去に提案した手法[39]を用いて構成される．我々のスケジューラはいくら巡回する URL 数が多くても，URL1 つに対するスケジュールに必要な計算量は変わらない O(1)であるスケジューラを実装している．URL にはドメイン名が異なっても IP が同じ場合が存在する．我々のモジュールは IP ごとに物理的にサーバが異なるものと判断し，IP によるスケジューリングを行い，任意のアクセス間隔を保証できる．詳しくは論文[39]を参照されたい

5. 1. 6　ダウンローダ

本モジュールはダウンロード処理を担う．ダウンローダは HTML 文書と画像や動画などのバイナリデータを，拡張子を利用して区別してダウンロードする．HTML 文書は次に説明するパーサに送る．

5. 1. 7 HTML パーサ

ダウンロードした HTML 文書からリンク先の URL を抽出する．このとき，同時に head タグ内の meta タグに記述可能な，クローラに対するアクセス許可指定も抽出することができる．また，HTML 文書とともに画像ファイルの URL も抽出する設計となっている．

5. 1. 8 ファイル出力

本モジュールはダウンロードした HTML や画像などのデータをファイルに出力する．本モジュールは，合計 4GB までの HTML および画像をそれぞれ 1 つのファイルにまとめて保存するように設定してある．これは，データ解析時にハードディスクからシーケンシャルな読み出しを実現するためである．

5. 2 QueueLinker[41]との連結

提案するパイプライン型の Web クローラを実現させる際に，パイプライン型処理の分散処理フレームワークである QueueLinker[41]を用いている．QueueLinker のフレームワークを用いる際にユーザに必要なタスクは，個々のモジュールの実装とモジュール間の接続関係を定義することである．6. 3 節の実験では 5. 1 節で説明した個々のモジュールを実装し，QueueLinker 上で Web クローラを動作させている．なお，今回の実験は分散環境下では行われていない．

5. 3 まとめ

本章では検証用に作成したパイプライン型の Web クローラで用いられているモジュールがどのように構成されているかを説明した．FIFO キューとメッセージでタスクをやりとり

することで各モジュールの依存関係を無くすことが可能となり，モジュールのカスタマイズ性に非常に優れている．また，各モジュールを流れる WebStreamData のデータを流すことで，エンドユーザに任意のモジュール間データを容易に提供することができる．

第6章　評価実験

本章では，第 4 章で提案したパイプライン型の Web クローラが 4. 1 節で掲げた要求条件をクリアできるかを確認するために行った実験について説明する．6. 2 節の実験では提案するパイプライン型の Web クローラの動作確認と性能評価を，6. 3 節ではモジュールの追加・変更・削除が可能なことの確認に加え，任意のモジュール間メッセージをエンドユーザへストリームとして提供できることの確認を行っている．

6. 1 実験概要

実際の Web 空間は，新たな Web ページの出現・リンク構造の変化・Web ページのアップデートといった変化が日々取り巻いている．しかし，いくつかの基本的な Web クローリングのテストを行い結果の比較をする際には同じ Web 空間に対して Web クローリングのテストを試行することが望ましい. そのため, 6. 2 節の基本的な実験に際しては, WebGraph[40] のプロジェクトで提供されているデータを基に仮想 Web 空間を 1 つの Web サーバ上に構築した．表 2 に今回の実験で用いたデータセットを記し表 3 に Web サーバとして用いたマシンのスペックを記載する．表 2 中の uk-2007-05 のデータセットは 2007 年 5 月時点のイギリスの Web 空間が記録されており，URL と Web のグラフ構造を取得することができる．

表 2 実験データ

Data Set	URL	Unique Domain	link
uk-2007-05	105,896,555	114,506	3,738,733,648

表 3 実験環境

	CPU	メモリ	考慮事項
実験環境 1	Quad-Core Operation(tm) Processor 8381 HE2.49GHz×16	256GB	ルーターボトルネックのため LocalHostアクセス

仮想 Web サーバを構築する際には，GNU libmicrohttpd[42]のライブラリを用いており，実験環境 1 のマシン上で動作している．GNU libmicrohttpd は C++で記述されており，GNU libmicrohttpd が動作しているサーバに HTTP プロトコルによる通信を行うことができる．なお，本実験ではルータによるボトルネックを回避するため，仮想 Web サーバに localhost としてアクセスすることで Web クローリングテストを行っている．仮想 web サーバはアクセスされた URL に対し，その URL が持つリンク先一覧 URL を uk-2007-05 のグラフ構造に従ってレスポンスするように構成されている．

6. 2 評価実験動作確認とパフォーマンステスト

6. 1 節で述べたように，本節の実験は uk-2007-05 のデータセットに基づいた仮想 Web 空間に対して行う．本節では 2 パターンの簡単な実験結果を記載すると共に，本実験で用いたクローラの特性と性能を説明する．また，今回の仮想 Web サーバの実験で解決できなかった課題点についても 6. 2. 3 節と 6. 2. 4 で考察する．

なお，今回の仮想 Web サーバをクローリングする実験では 5. 1. 3 節で記述した robots.txt 処理のモジュールに関しては何の処理もしないように変更を施してある．

6. 2. 1 Web クローラの基本動作確認

本節では提案した Web クローラの基本動作を確認するために行った実験について説明する．我々が過去に提案したスケジューリング手法[39]によって，同一 IP アドレスに対するクローリング間隔を最低 30 秒間に設定した場合のクローリング結果を図 10～図 14 に示す．各図は，それぞれ以下の対応となっている．なお，シードに用いたデータはアウトリンク数が多かった 50 個の URL を使用している．

図 10 クローリング経過時間とダウンロードした HTML の数の関係

図 11 クローリング経過時間と検出したドメイン数の関係

図 12 クローリング経過時間と平均ダウンロード速度[pages/s]

図 13 クローリング経過時間とスケジューラにストアされた URL 数の総和

図 14 クローリング経過時間と平均ダウンロード速度[pages/s]

シードに含まれているユニークなドメイン数が少ないため，クローリング開始直後のダウンロード速度は非常に遅いことが図 12 から読み取れる．また，時間の経過と共に新たなユニークなドメインを発見する(図 11)に連れ，アクセス速度が上昇する．また，正しく 30 秒周期でダウンロードを試行していることが図 10 から確認できる．また，図 13 は多くの URL は同一ドメインであるため，スケジューラの中でストアされる StreamWebData が増加していることを示している．図 14 は，今回の実験においてはクローリング開始後 12000[s]がダウンロード速度のピークを迎えていることを示しており，その際のダウンロード速度はおよそ 200url/s であり，以降クローリングが完了したドメインが増加するに連れて徐々にダウンロード速度が減少している様子を表している．これらの実験により，提案手法によるパイプライン型 Web クローラが，意図したモジュールの実装通り正しく動作することを確認できた.

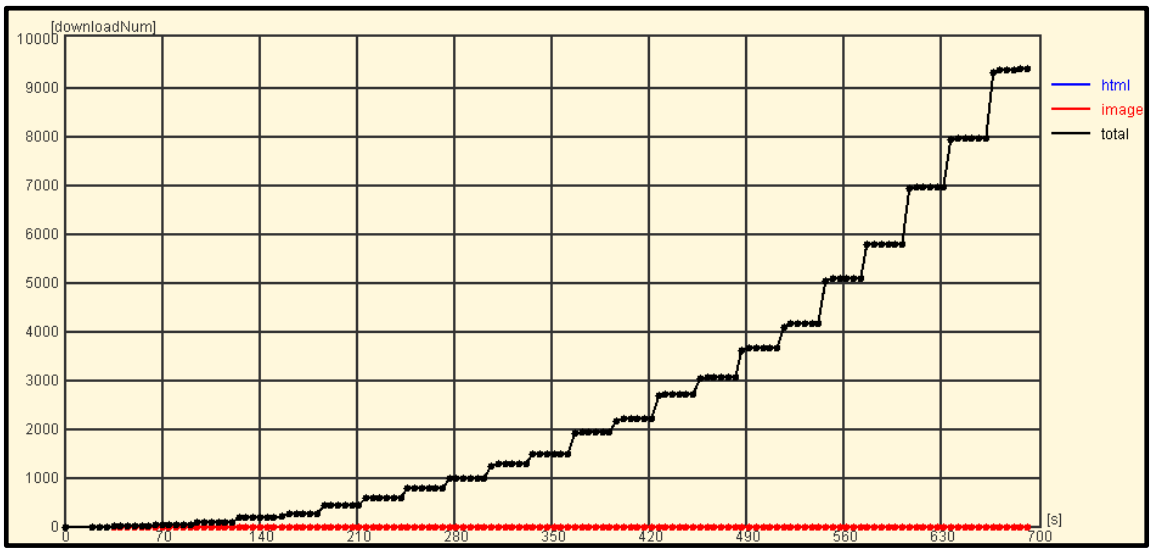

図 10　クローリング経過時間とダウンロードした HTML の数の関係

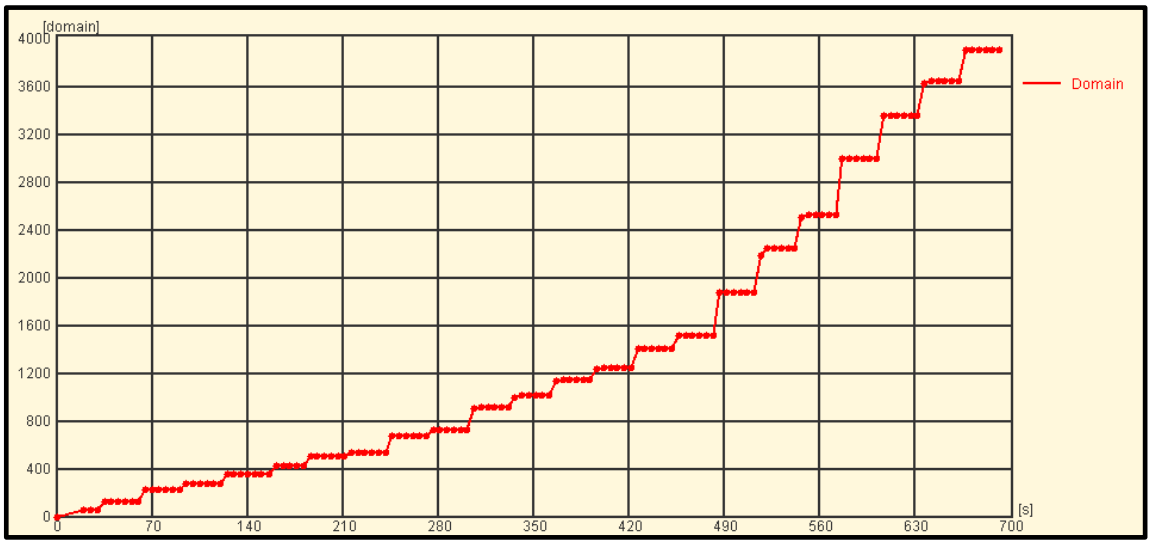

図 11　クローリング経過時間と検出したドメイン数の関係

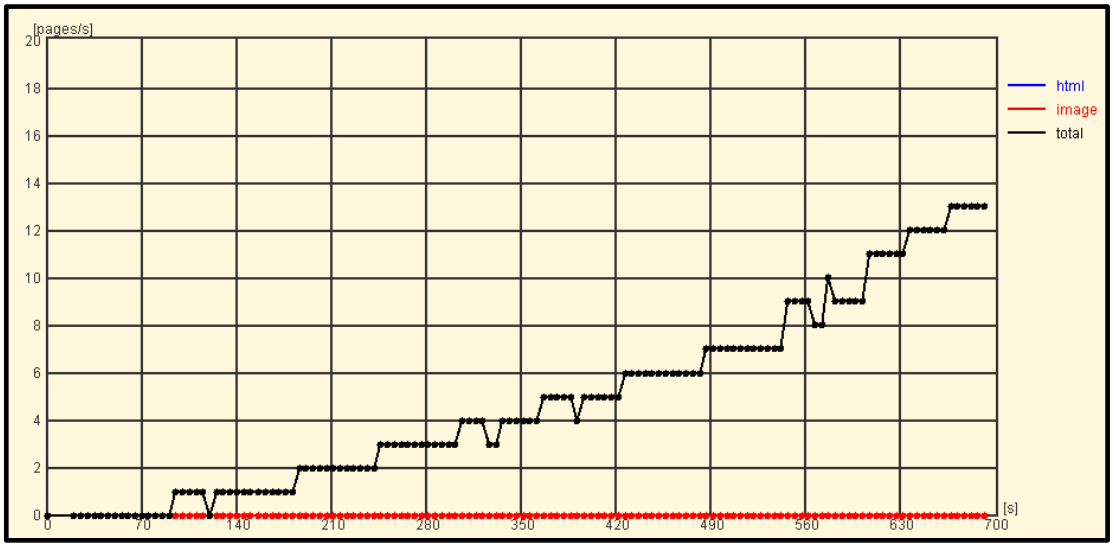

図 12　クローリング経過時間と平均ダウンロード速度[pages/s]

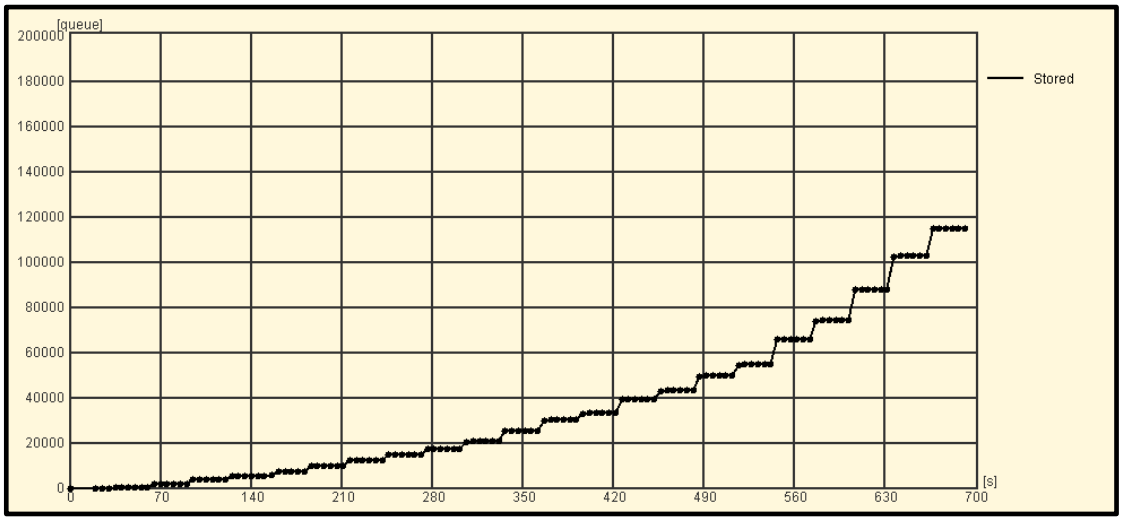

図 13 クローリング経過時間とスケジューラにストアされた URL 数の総和

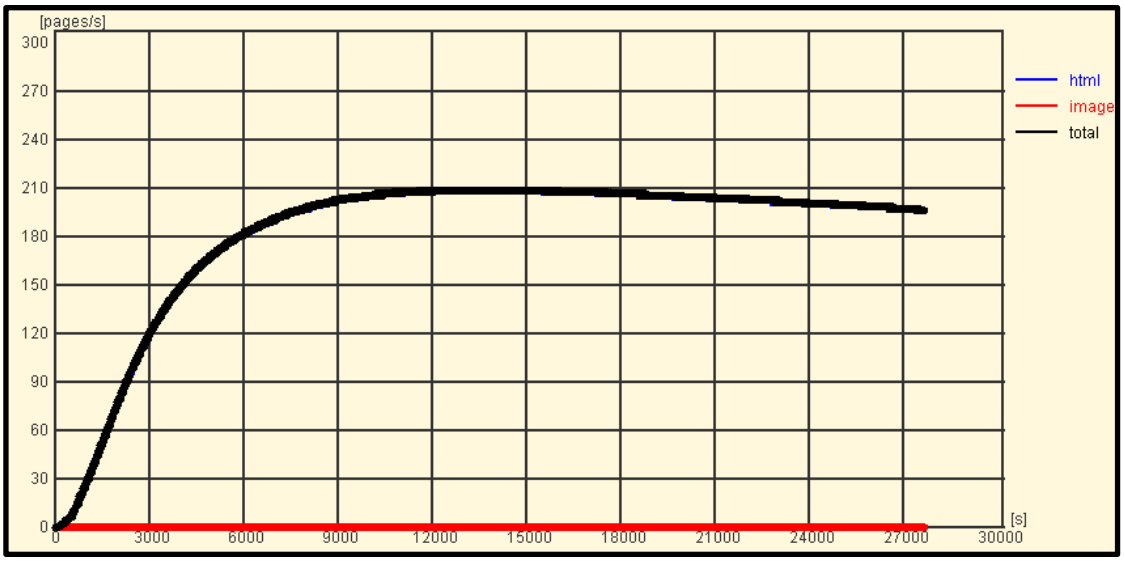

図 14　クローリング経過時間と平均ダウンロード速度[pages/s]

6. 2. 2 Web クローラの性能確認

本節では，実験で作成した Web クローラが高速且つスケールすることを確認するために仮想 Web サーバに対して行った実験について説明する．6. 2. 1 節の実験では同一 IP アドレスに対して最短でも 30 秒間のアクセス間隔を保証して実験を行ったが，本節の実験では 50ms という時間を設定して実験を行っている．また，シードに使用する URL 数もアウトリンク数の多い 10,000 の URL を使用している．これらの設定に変更したクローリング結果を図 15～図 19 に示す．各図は，それぞれ以下の対応を示している．

図 15 クローリング経過時間とダウンロードした HTML の数の関係

図 16 クローリング経過時間と平均ダウンロード速度[pages/s]

図 17　HTML のダウンロード速度[KBps]

図 18 各モジュールにストアされている Queue のサイズ

図 19 クローリング経過時間とスケジューラにストアされた URL 数の総和

図 16 より, 平均して 2000-3000[url/s]のペースでダウンロードができていることが分かる．この値は IRLBot[18]のダウンロード速度と非常に近い値となっている．収集する URL の規模やマシンの性能差から一概に述べることは言えないが，IRLBot に近いパフォーマンスを期待できる．その際の HTML テキストのダウンロード速度は 12-14[MBps]程度となっている(図 17)．今回収集した HTML のテキストデータはリンク情報のみという比較的小さなデータであるため，ネットワーク通信の多くの割合のパケットをプロトコル通信に必要なデータが占めている．そのため，URL の平均ダウンロード数に対するテキストデータのダウンロード速度の値が比較的小さくなっている．また，図 18 は各モジュールに蓄えられているキューの大きさが示されており，今回の実験により名前解決とダウンローダがボトルネックとなる可能性が高いことが読み取れる．また，各図の 5500s～5800s 付近で確認できる空白の期間は，Java のガーベッジコレクションによってプログラムが一時的に停止している状態である．

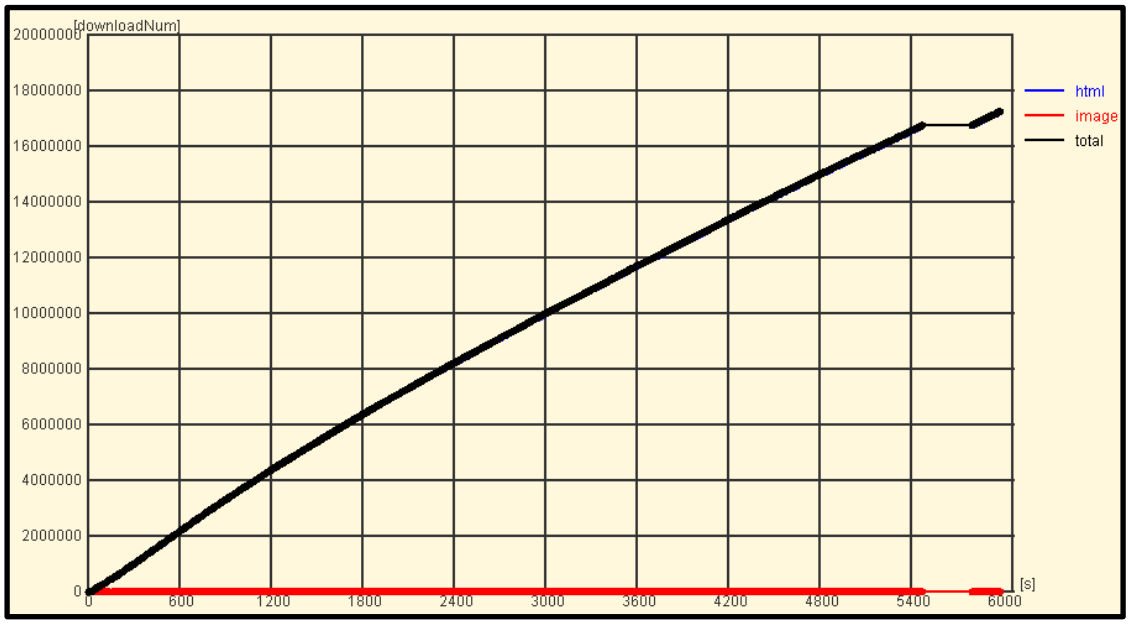

図 15 クローリング経過時間とダウンロードした HTML の数の関係

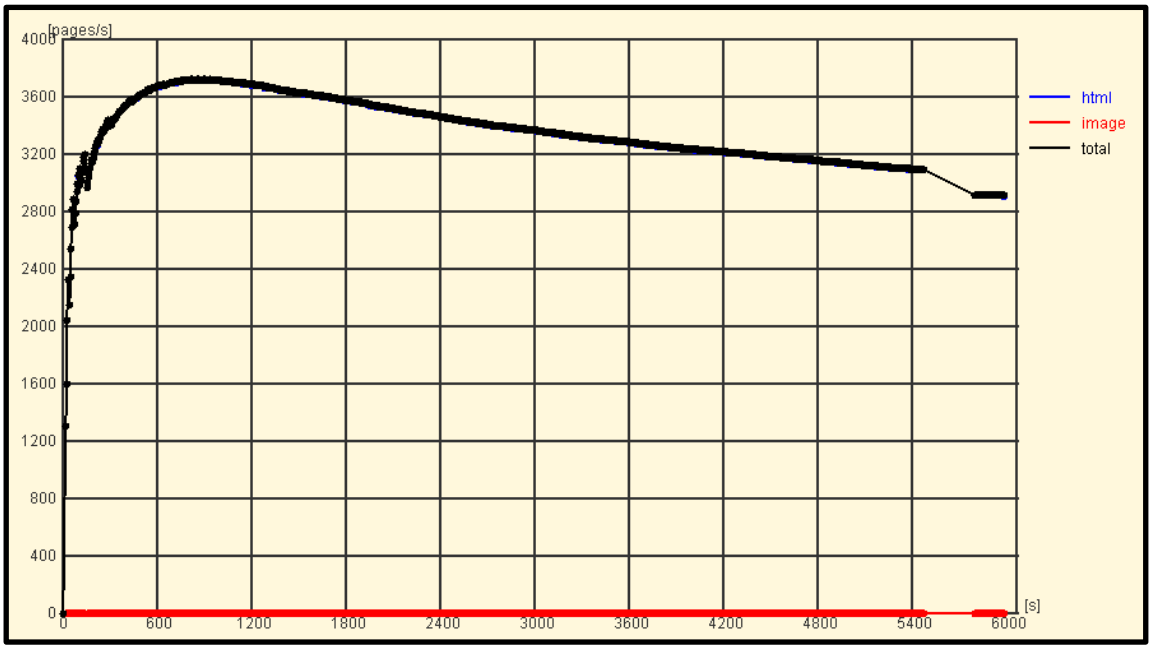

図 16 クローリング経過時間と平均ダウンロード速度[pages/s]

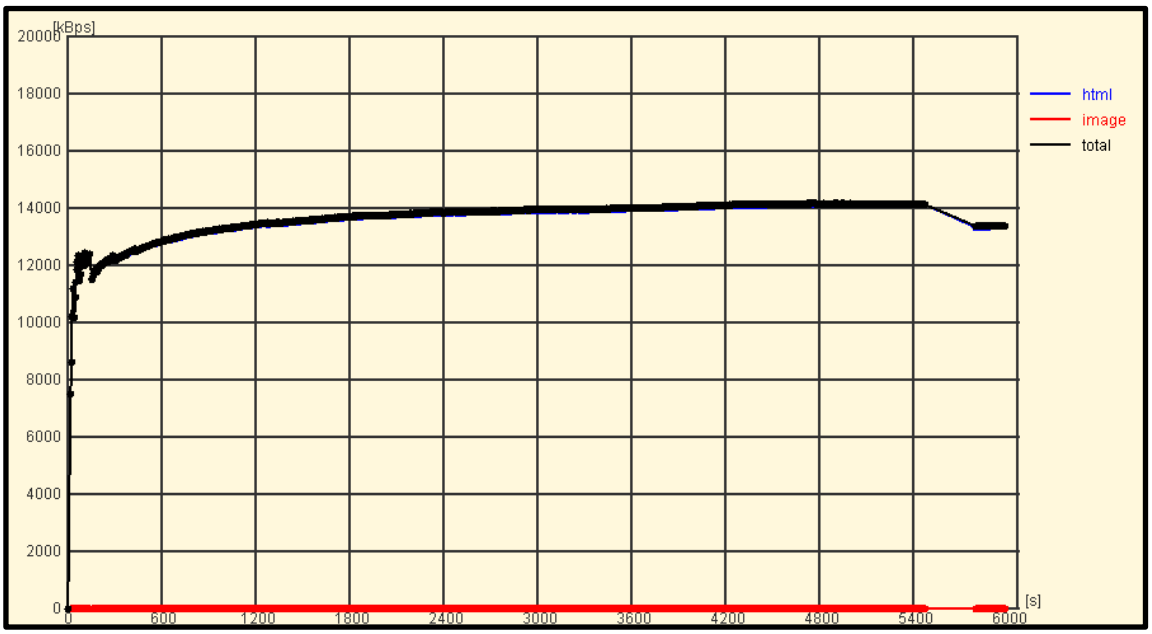

図 17 HTML のダウンロード速度[KBps]

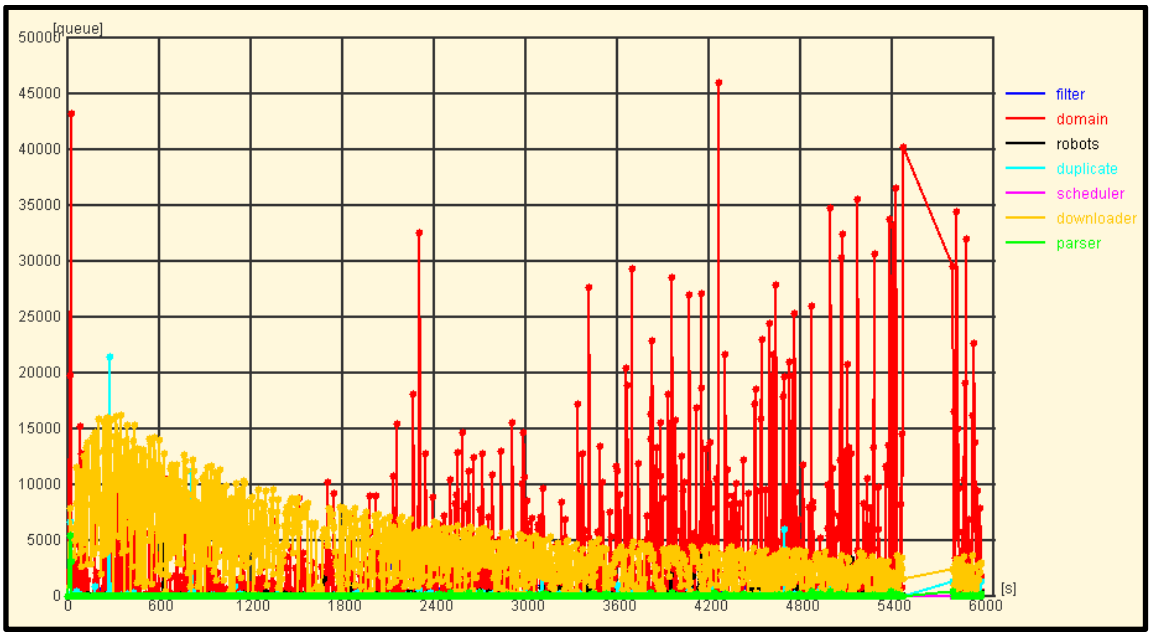

図 18 各モジュールにストアされている Queue のサイズ

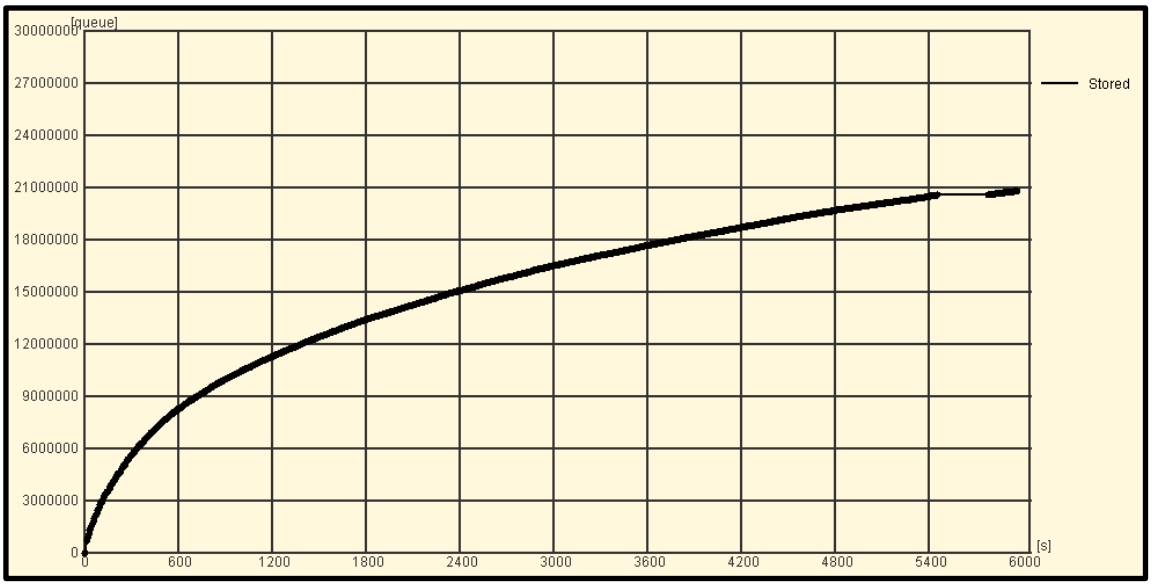

図 19 クローリング経過時間とスケジューラにストアされた URL 数の総和

6. 2. 3 現状の Web クローラの課題点

本実験で，いくつかの解決すべき問題が判明したので，ここに報告をする．

1. ガーベッジコレクションが動いた際に，プログラムが一定時間停止する

2. ルータが脆弱な場合，ネットワーク通信の大きなボトルネックとなる恐れがある

1 に関しては，Java のガーベッジコレクションに適切なオプションを設定することで軽減することができる場合もある．しかし，オプションを誤ると，普段のガーベッジコレクション処理に CPU 時間を大きく取られてしまい，性能が大幅に低下する可能性もある．

6. 2. 4 仮想 Web サーバの課題点

本実験を行うにあたり，URL をエンコードした際に特殊文字を含むいくつかの URL に対して，ダウンロード処理が仮想 Web サーバにアクセスできないという問題点が解決しき

れていない．例えば，半角スペースは”%20”，改行(Line Feed)は”%0A”に変換される．中には対応しきれていない特殊文字が存在しており，全体で一億強ある Web ページ全ての URL にアクセスできず，クローリングできる仮想 Web 空間が全体の 10%-15%程度縮小される現象が課題として残っている．対応できていない特殊文字の対応を表 4 に記載する．

表 4　未対応特殊文字一覧

文字	URLエンコード	文字	URLエンコード	文字	URLエンコード	文字	URLエンコード
バックスペース	%08	¥	%A5	À	%C0	à	%E0
タブ	%09	¦	%A6	Á	%C1	á	%E1
改行(Line Feed)	%0A	§	%A7	Â	%C2	â	%E2
復帰(Carriage Return)	%0D	«	%AB	Ã	%C3	ã	%E3
スペース	%20	¬	%AC	Ä	%C4	ä	%E4
!	%21	¯	%AD	Å	%C5	å	%E5
″	%22	°	%B0	Æ	%C6	æ	%E6
#	%23	±	%B1	Ç	%C7	ç	%E7
$	%24	ª	%B2	È	%C8	è	%E8
%	%25	`	60%	É	%C9	é	%E9
&	%26	{	%7B	Ê	%CA	ê	%EA
’	%27	¦	%7C	Ë	%CB	ë	%EB
(	%28	}	%7D	Ì	%CC	ì	%EC
)	%29	~	%7E	Í	%CD	í	%ED
*	%2A	¢	%A2	Î	%CE	î	%EE
+	%2B	£	%A3	Ï	%CF	ï	%EF
,	%2C	¥	%A5	Ð	%D0	ð	%F0
;	%3B	¦	%A6	Ñ	%D1	ñ	%F1
<	%3C	§	%A7	Ò	%D2	ò	%F2
>	%3E	«	%AB	Ó	%D3	ó	%F3
[	%5B	¬	%AC	Ô	%D4	ô	%F4
¥	%5C	¯	%AD	Õ	%D5	õ	%F5
]	%5D	°	%B0	Ö	%D6	ö	%F6
^	%5E	±	%B1	Ø	%D8	÷	%F7
_	I	ª	%B2	Ù	%D9	ø	%F8
`	0.6	¸	%B4	Ú	%DA	ù	%F9
{	%7B	µ	%B5	Û	%DB	ú	%FA
¦	%7C	»	%BB	Ü	%DC	û	%FB
}	%7D	¼	%BC	Ý	%DD	ü	%FC
~	%7E	½	%BD	Þ	%DE	ý	%FD
¢	%A2	¿	%BF	ß	%DF	þ	%FE
£	%A3					ÿ	%FF

6. 3 カスタマイズ性とストリームデータ提供に関する実験

6. 2 節では仮想 Web 空間をクローリングすることで，作成したパイプライン型 Web クローラが正しく動作することを確認した．本節では，WWW に対してクローリングを行い，モジュールの変更と追加に対する実験及び，エンドユーザにクローリングした Web データをリアルタイムストリームとして提供する実験について述べる．また，本節の実験は QueueLinker によるフレームワークを用いて実験を行っている．

6. 3. 1 モジュールの変更

本節では，モジュールの変更が容易であり，提案する Web クローラがソフトウェアアーキテクチャの観点から粗な結合であることを確認するために行った実験について説明する．なお, 本実験でのモジュールの変更は 5. 1. 4 節で説明した重複削除の手法を変更している．図 20 の紫色で示された部分が，重複削除モジュールを変更した様子を表している．

5. 1. 4 節で説明した重複削除モジュールは，最初に LRU キャッシュを用いたアルゴリズムで, 対象の URL を最近訪れたことがないかを高速に検出する. そしてその後に BloomFilter を用いて URL の重複を高速かつメモリを節約しながら発見する手法を取っていた. しかし, この手法では一部の重複していない URL が BloomFilter によって重複していると判断されてしまう可能性がある[38]．そこで確実な URL の重複を検出するため，Set を用いて重複除去をするモジュールに変更を施した．重複削除の手法の変更に伴って，他モジュールにはソースコードに一切の変更は加えられておらず，正しくそれぞれのモジュールの役割を果たしていることを確認した．

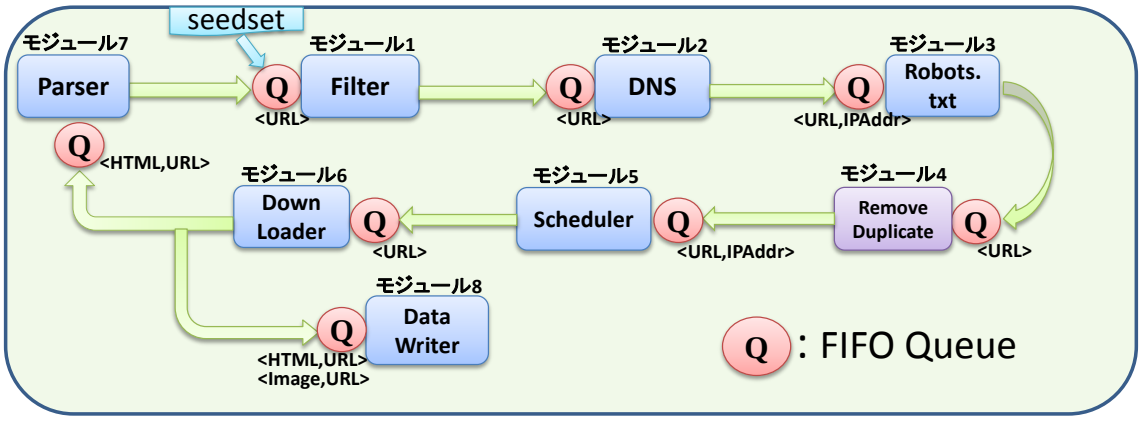

図 20　重複削除モジュールのストラテジーの変更

6. 3. 2　モジュールの追加

本節では，モジュールの追加を容易に行えることを確認するための実験について説明する．提案する Web クローラは新たなモジュールを追加する際に，キューの接続関係を変更することにより実現される．本実験では，指定したワードを含む Web サイトと，その近辺の Web サイトを優先して収集できるように，指定したワードを含んだ Web ページのみをフィルタするモジュールを追加した．その様子を図 21 に示す．図 21 の赤色で示された矢印はキューを介したモジュールの接続関係が変更された部分を示しており，紫色部分で示されたモジュール 9 が実際に新たな追加されたモジュールである．モジュール 9 の Word Filter は HTML テキストに指定されたワードが含まれていた場合のみ Parser にその HTML を通過させるフィルタリング機能を持つモジュールである．本実験により，キューの接続関係を変更することで，新たなモジュールの追加を行える設計となっていることを確認した．なお，QueueLinker 上で moduleA,moduleB 間に新たな moduleC を挿入する際の疑似コードを図 22 に記載する．図 22 中の 6 行目の赤色で示されたコードを 7,8 行目に示された青色のコードに変更するだけでキューの接続関係を変更することができるため，QueueLinker を用いた際にはキューの接続関係をわずか数行で変更可能である．

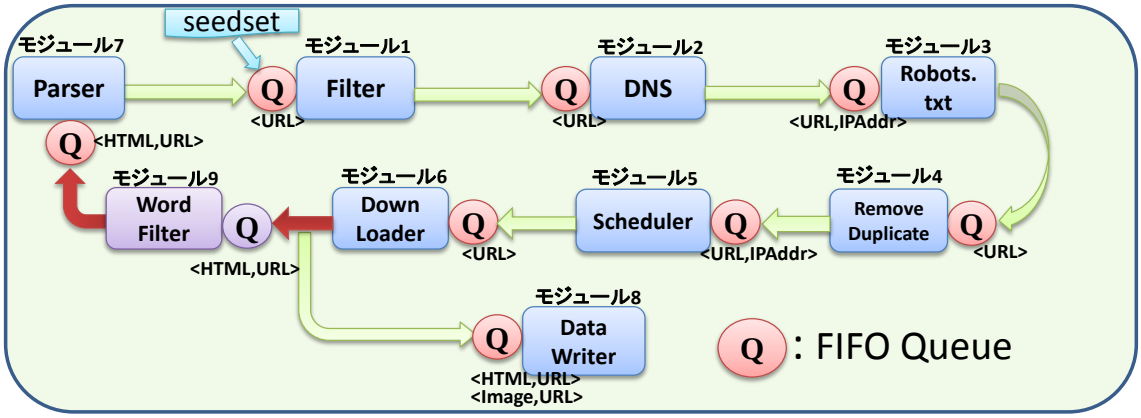

図 21　指定したワードを含む Web ページをフィルタするモジュール

Algorithm : 各モジュールの処理

```
1. LogicalGraph graph;   //モジュールの接続関係を表すグラフ
2. LogicalVertex moduleA; //モジュール A
3. LogicalVertex moduleB; //モジュール B
4. LogicalVertex moduleC; //追加するモジュール C
5.
6. Graph.addLogicalEdge(moduleA, moduleB); //接続関係  moduleA⇒moduleB
7. Graph.addLogicalEdge(moduleA, moduleC); //接続関係 moduleA⇒moduleC
8. Graph.addLogicaEdge(moduleC, moduleB); //  接続関係  moduleC⇒moduleB
```

図 22　QueueLinker によるキューの接続関係定義

6. 3. 3　ユーザへの任意モジュール間データのストリーム送信

本節では，提案する Web クローラが任意のモジュール間のストリームをエンドユーザアプリケーションに提供できることを確認するために行った実験について説明する．本実験では 2 つのエンドユーザアプリケーションを想定した．1 つはドメイン名と IP アドレスの対応表を作成する DNS 対応表のアプリケーション，1 つは Web 上の単語の出現頻度をカウントする，単語出現頻度カウントのモジュールである．

DNS 対応表を作成するアプリケーションは名前解決モジュールの後から流れるストリー

ムデータを受け取る．単語出現頻度カウントモジュールは，ダウンロードモジュールから，実際にダウンロードされた HTML データを受け取ることにより実現される（図 23）．本実験では，それぞれのモジュール間データをエンドユーザアプリケーションのリッスンポートに向けて送信することにより，適切なモジュール間データをリアルタイムストリームとして取得することが可能であることを確認した．

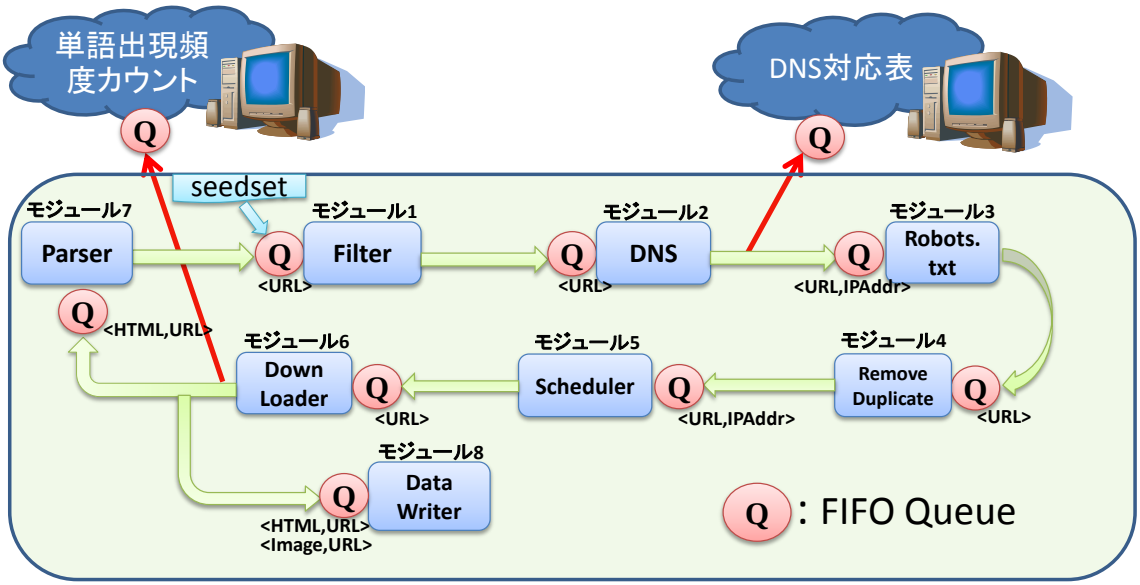

図 23　モジュール間を流れるストリームデータの利用

6. 4 Future Work

より成熟したシステムにするためにはいくつかの課題が残っている．1 つはクローリングを重ねるにつれ，基本的に Web クローラを流れるデータは増加し続けるため，メインメモリにデータが収まりきらなくなる課題が挙げられる．その課題に関しては，各モジュールが自分の FIFO キューのサイズが大きくなりすぎたと判断した場合にハードディスクにデータを出力することによって対処する手法が考えらえる．あるいは，ダウンロードの速度を調整することにより，Web クローラ全体を流れるデータを抑止する手法も考えらえる．また，QueueLinker を用いた 6. 3 節の実験では分散環境下で Web クローリングを行っていない．今後，よりスケールする Web クローラを作成するためには，分散環境下においての実行も保証しなければならないことが課題として残っている．

また，いくつかのモジュールにおいて不完全な部分が残っている．例えば，5. 1. 3 節の robots.txt の処理モジュールにおいて，再収集する機構が実装されていない．また，サイトマップに記載された情報を有向活用する機構も存在していない．さらに，現在 ipv6 への移行が一部で進みつつあるが，ipv6 への対応は実装されていない．最後に，今回の実験では Web クローラを流れるストリーム形式のデータ構造を StreamWebData として定義したが，JSON や XML 形式によって自由にデータを付加できる状態にしておいた方が，より柔軟な Web クローラを作成することに結びつくことも考えられる．

6. 5 まとめ

本章では，6. 1 節にて実験概要について述べ，6. 2 節にて提案する Web クローラの動作確認と性能評価を，6. 3 節では(1).モジュールの変更・追加が可能なこと (2).クローラのモジュール間データを，リアルタイムストリームとして受信可能．という二点を満たしていることを確認し，6. 4 節にて今後より実用的な Web クローラに発達するための課題についてまとめた．

第7章　おわりに

従来の Web クローラはモジュールの変更・追加・削除といった必要性が生じた際に，他のモジュールに変更の影響が及ぶ可能性があった．また，実際に Web クローラで収集したデータをエンドユーザに提供する手段はデータベースの検索による従来の検索エンジン型の構成である種類が多かった．我々が提案するパイプライン型の Web クローラは，全てのモジュールが完全独立な設計である．そのため，あるモジュールの変更，あるいは追加・削除が，他のモジュールへ機能的に一切の影響を与えないことを明確に保証する．また，任意のモジュール間を流れるストリームデータをエンドユーザに提供することが可能である．そのため，よりリアルタイムで適切なストリームデータを容易にエンドユーザに提供できる構成となっている．本研究では実際にモジュールの変更・追加に対する実験を行うことにより，提案する Web クローラがソフトウェアアーキテクチャの観点から租な設計となっていることを確認した．また，任意のモジュール間データをリアルタイムストリームとして活用できることを確認するため，DNS 対応表を作成するアプリケーション，指定された単語をカウントするアプリケーションへモジュール間データを流れるストリームをリアルタイムで提供できることを確認した．

謝辞

本研究を行うに当たり，多くの御指導，御助言を頂いた山名早人教授，日頃から様々なアドバイスを頂いた上田高徳助手，並びに日頃の研究生活で切磋琢磨した同期の皆様，先輩，後輩の皆様に厚く御礼申し上げます.

また，本研究の一部は，文部科学省次世代 IT 基盤構築のための研究開発「Web 社会分析基盤ソフトウェアの研究開発」(多メディア Web 解析基盤の構築及び社会分析ソフトウェアの開発)」によるものである.

参考文献

[1] Y. Morimoto, Y. Taguchi, and T. Naemura, "Automatic colorization of grayscale images using multiple images on the web", In Proc. of SIGGRAPH 2009: Posters, USA, Aug. 3-7. 2009.

[2] 黒木さやか, 山名早人, 立石健二, 細見格, "アンカーテキストとリンク構造を用いた同義語抽出手法", DEIM 2010, 兵庫, Feb. 28 - Mar. 2 2010.

[3] Web Archive: http://www.archive.org/. (2012.1.29 参照)

[4] データセクション株式会社: http://www.datasection.co.jp/ (2012.1.29 参照)

[5] lucene: http://lucene.apache.org/. (2012.1.29 参照)

[6] nutch: http://nutch.apache.org/. (2012.1.29 参照)

[7] solr: http://lucene.apache.org/solr/. (2012.1.29 照)

[8] YaCy - The Peer to Peer Search Engine: Home: http://yacy.net/. (2012.1.29 参照)

[9] 80legs: http://www.80legs.com/ (2012.1.29 参照)

[10] H. Dong, F.K. Hussain and E. Chang, "Focused crawling for automatic service discovery, annotation, and classification in industrial digital ecosystems", Industrial Electronics, IEEE Transactions on, vol.58, pp.2106-2116, 2011.

[11] H. Liu, E. Milios, and J. Janssen. "Probabilistic models for focused web crawling", In Proc. of WIDM, pp.16–22, 2004.

[12] T.J. Fu, A. Abbasi, and H. Chen, "A Focused Crawler for Dark Web Forums", Journal of the American Society for Information Science and Technology (JASIST),2010.

[13] R.A. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez,"Crawling a country: better strategies than breadth-first for web page ordering", In Proc. of the World Wide Web Conference, pp.864-872, 2005.

[14] D. Fetterly, N. Craswell, and V. Vinay,"The impact of crawl policy on web search effectiveness ", In Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 580–587, 2009.

[15] Streaming API: Twitter Developers: https://dev.twitter.com/docs/streaming-api(2012.1.29 参照)

[16] Jonathan M. Hsieh, Steven D. Gribble and Henry M. Levy, "The Architecture and Implementation of an Extensible Web Crawler", In Proc. of the 7th USENIX conference on Networked systems design and implementation, 2010.

[17] The Web Robots Pages: http://www.robotstxt.org/.(2012.1.29 参照)

[18] HLee, H.-T., Leonard, D. Wang, X. and Loguinov, D. "Irlbot: scaling to 6 billion pages and beyond", In: 17th International Conference on World Wide Web, pp. 427–436, 2008

[19] J. Cho, H. Garcia-Molina, T. Haveliwala, W. Lam, A. Paepcke, S. Raghavan and G. Wesley, "Stanford WebBase Components and Applications", ACM Transactions on Internet Technology, vol.6, No.2, pp. 153-186, May 2006.

[20] Official Google Blog: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html (2012.12.16 参照)

[21] Allan Heydon and Marc Najork. Mercator: A Scalable, Extensible Web Crawler, World Wide Web, Volume 2, No.4, pp.219-229, December 1999.

[22] Dennis Fetterly, Mark Manasse, and Marc Najork. “Spam, damn spam, and statistics”, In Proceedings of the Seventh International Workshop on the Web and Databases (WebDB), 2004

[23] C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. “Know your neighbors: web spam detection using the web topology”, In Proc. SIGIR ’07, pages 423–430, New York, NY, USA, 2007.

[24] 田村孝之, 喜連川優, “大規模 Web アーカイブ更新のための階層的スケジューリング手法”,情報処理学会論文誌データベース, Vol. 2, No. 3, pp. 67-75, Sep. 2009.

[25] 田村孝之, 喜連川優, “多周期的更新アクセスに適した二次記憶管理技法”, 電子情報通信学会論文誌 vol.J93-D No.6 p805-815, 2010.

[26] D. Denev, A. Mazeika, M. Spaniol, and G. Weikum, “SHARC: Framework for quality-conscious web archiving”, PVLDB, 2(1):586–597,2009.

[27] Madaan, R., Dixit, A., Sharma, A.K., and Bhatia, K.K, “A Framework for Incremental Hidden Web Crawler”, In International Journal on Computer Science and Engineering, V ol. 02, No. 03, pp. 753-758. 2010.

[28] Clam AntiVirus: http://www.clamav.net/lang/en/ (2012.1.29 参照)

[29] D. Eichmann, "The RBSE Spider - Balancing Effective Search Against Web Load", In Proc. of 1st World Wide Web Conference, pp. 113-120, Switzerland, May 25-27. 1994.

[30] J. Edwards, K. McCurley and J. Tomlin, “An Adaptive Model for Optimizing Performa nce of an Incremental Web Crawler”, In Proc. of the 10th WWW, China, pp.106-113, May. 2001.

[31] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. “Ubicrawler: A s calable fully distributed Web crawler”, In Proc. AusWeb02. The Eighth Australian World Wide Web Conference, 2002.

[32] V. Shkapenyuk and T. Suel, “Design and Implementation of a High-Performance Distrib uted Web Crawler”, In Proc. of the 18th ICDE, USA, Feb. 2002.

[33] A. P. Tao, H. Fengling, Z. Wanli “A new Framework for Focused Web Crawling” , Wu han University Journal of Natural Science (WUJNS), 2006.

[34] A. Singh, M. Srivatsa, L. Liu and T. Miller, “Apoidea: A Decentralized Peer-to-Peer Ar chitecture for Crawling the World Wide Web”, In Proc. of SIGIR Workshop on Distribu ted IR, Canada, pp. 126-142, Aug 2003.

[35] Qing-Cai Chen, Xiao-Hong Yang amd Xiao-Long Wang; “A Peer-to-peer based passive web crawling system”, Machine Learning and Cybernetics (ICMLC) p.1878 - 1883 , Jul y 2011.

[36] RFC1738: http://www.ietf.org/rfc/rfc1738.txt. (2012.1.29 参照)

[37] N. Askitis and R. Sinha, “HAT-trie: a cache-conscious trie-based data structure for strings”, In Proc. of 13th ACSC, pp. 97-105, Australasia, Jan. 30- Feb 2.

[38] 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 年 6 月.

[39] 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.

[40] Laboratory for Web Algorithmics:http://law.dsi.unimi.it/datasets.php(2012.1.29 参照)

[41] 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.

[42] GNU libmicrohttpd: a library for creating an embedded HTTP server: http://www.gnu.org/software/libmicrohttpd/(2012.1.17 参照)

付録 A　プログラムの説明

本実験中で用いたプログラムの実行方法と簡潔な説明をここに記載する.

Ⅰ　仮想 Web サーバ　(mach.yama.info.waseda.ac.jp にデータ保存)

1.　libmicrohttpd(http://www.gnu.org/software/libmicrohttpd/)をマシンにインストール

・インストール済み

2.　ソースコード実行方法

I.　cd /home/work/uchida/mhttdpd

II.　g++ -o dummyserv –O2 –L/usr/local/lib –lmicrohttpd mhttpd.cpp

III.　export LD_LIBRARY_PATH=/usr/local/lib

IV.　./dummyserv18080(←ポート番号は自由に変更可能)

3.　使用する URL，リンク先データ保存場所

WebGraph (http://law.dsi.unimi.it/datasets.php)に公開されている uk-2007-05 を使用

URL；　/work/uchida/data/decodedURLs

リンクデータ：/work/uchida/graphlinks

4.　仮想 Web サーバの対象 URL へ HTTP でアクセスする方法

ex. http:// yama.info.waseda.ac.jp/index.html　を仮想 Web サーバに登録した場合

http://HOSTNAME:18080/yama.info.waseda.ac.jp/index.html　にてアクセス可能

5.　Web クローラからアクセスする方法

・module.download の Downloader クラス

getOperation(URL url, FileType type)　メソッドを変更

url に HOSTNAME:18080 を挿入

実行例(HOST = mach.yama.info.waseda.ac.jp)

http://winki.co.uk/company/main.htm を仮想 Web サーバに構築し，アクセスした場合

アクセス先 URL：

http://mach.yama.info.waseda.ac.jp:18080/winki.co.uk/company/main.htm

レスポンス例(リンク先 URL 一覧)

http://winki.co.uk/company/contact.htm
http://winki.co.uk/company/database.htm
http://winki.co.uk/company/ecomerce.htm
http://winki.co.uk/company/graphics.htm
http://winki.co.uk/company/webdesign.htm
http://www.skwigly.co.uk/
http://www.around.co.uk/mugshot/default.asp
http://www.winki.co.uk/
http://www.around2.co.uk/
http://www.schoolzone.co.uk/
http://www.buymore.co.uk/
http://www.shopsimple.co.uk/
http://www.around.co.uk/
http://www.around.co.uk/porsche-forums/

Ⅱ　Web クローラ

リポジトリ：

https://fs.yama.info.waseda.ac.jp/svn/repos/share/Crawler

./conf/より各モジュールのオプションを設定可能

主な設定ファイルとその内容

- core.properties.xml
  - シードに用いるファイル定義
  - テストモードか否かを決定
- blacklist.list, whitelist.list
  - 遮断/通過するドメインを設定
  - 先頭に'!'にてコメントアウト
- scheduler.properies.xml
  - 各 IP に対するアクセス間隔の設定が可能
- file.properies.xml
  - ファイル出力の可否を設定可能
  - 出力時の 1 つのファイルサイズを設定可能
- logger.properties.xml
  - データベース先の設定を可能
- robots.txt.properies.xml
  - 更新間隔を設定可能(未実装)
- download.properties.xml
  - タイムアウトを設定可能
- domain.properties.xml
- duplicate.check.properies.xml

いずれも各モジュールのスレッド数設定可能

仮想空間 Web クローラ

リポジトリ：

https://fs.yama.info.waseda.ac.jp/svn/repos/share/uchida/DummyCrawler

・monitor/DataSendClienet.java にて相手サーバ，ポートを指定

GUI プログラム

リポジトリ：

https://fs.yama.info.waseda.ac.jp/svn/repos/share/uchida/CrawlerGUI2

・gui/GUIMain.java から起動

・server/DataRecieveServer にて，受信するポート番号設定

HTML ダウンロード数，発見ホスト数，ダウンロード速度，各モジュールに蓄積されたキューのサイズ，使用メモリを監視可能

Ⅲ　QueneLinker 上での Web クローラの実行

リポジトリ

https://fs.yama.info.waseda.ac.jp/svn/repos/share/ueda/QueueLinkerCrawler/trunk

・main/QueueLinkerCrawlerJob.java を実行

・現在開発中

付録 B　業績リスト

国内研究会

- 打田 研二，高木 浩光，山崎 邦弘，山名 早人，“Winnyネットワーク上を流通するコンテンツの傾向と分析”，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.
- 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.
- 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.
- 佐藤亘，打田研二，山名早人“検索エンジンのヒット数に対する信頼性評価指標の提案とその妥当性検証”，情報研報,， Vol.2011-DBS-152, No.8, pp.1-8 July, 2011.

国内シンポジウム

- 上田高徳，片瀬弘晶，森本浩介，打田研二，山名早人，“QueueLinker: Distributed Producer/Consumer Queue Framework”，WebDB Forum（招待ポスター），Nov. 2009.
- 上田高徳， 打田研二，秋岡明香， 山名早人, “データストリーム処理におけるレイテンシ最小化と高可用性のためのオペレータ実行方法”, WebDB フォーラム 2011 (2011.11)

翻訳

- 打田研二，上田高徳 訳，Bertrand Meyer, Christine Choppy, Jorgen Staunstrup and Jan van Leeuwen 著，“視点：計算機科学分野における研究評価，” Communications of the ACM（日本語版），vol.9，no.1，pp.1-7，Jul. 2011.