修 士 論 文

稀にアクセスされるWebページを検出するシステムに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

立花 一樹

2012年2月

修 士 論 文

稀にアクセスされるWebページを検出するシステムに関する研究

指導教官　知念 賢一 特任准教授

審査委員主査　知念 賢一 特任准教授
審査委員　篠田 陽一 教授
審査委員　敷田 幹文 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

0810037 立花 一樹

提出年月: 2012 年 2 月

概 要

1991 年に World Wide Web が発明されてから、ホームページの閲覧だけでなく電子決済やニュースの閲覧、Web メールの利用など HTTP を利用した様々なサービスが実現されるようになった。Web を利用するユーザ層の拡大によってアクセス目的が多様化した結果、そのアクセス先となるインターネット上の Web コンテンツの情報量は莫大なものとなっている。しかし、Web ページのアクセス頻度とそれに基づくアクセス順位との関係は、一般的にべき乗則に従うことが知られている。このことから、大半のアクセスは一部の人気の高い Web ページへ集中していることが分かっている。これまでは、Web ページのランキング調査のように人気のある一部の Web ページの存在やそれへのアクセス分布は詳しく調べられてきたが、その他の膨大な数となるアクセス頻度が低い Web ページへのアクセスやその存在はほとんど知られていない。それらへの稀なアクセスの中には、単に人気のない Web ページへのアクセスだけでなく、例えば現在社会問題となっている Web を介して感染するマルウェアによる通信のような特異な目的の通信が含まれている可能性がある。

そこで、本研究はこれまで本格的に調査されることがなかった稀にアクセスされる Web ページ群へのアクセスを検出するために長期間に渡って観測可能なシステムを提案する。そして、これまで知られていない Web ページの存在やアクセスパターンを発見することで、これまで見過ごされてきた危険なアクセスを迅速に検出したり、Web の生態を解明する上で社会的に有用な知見を得ることができると考えている。研究を進めるにあたり、まずアクセス頻度とそれに基づく順位の関係から本研究が対象とする稀にアクセスされる Web ページ群の分布とそのデータ量を把握した。

次に、「稀な Web ページへのアクセス」を定義し、長期間に渡って実時間で検出可能なシステムを設計、試作した。検出システムには、様々な目的でアクセスされたあらゆる Web ページの URL を記録できる容量効率の高いデータ構造と、アクセス履歴に含まれる種々のパラメータの中から稀にアクセスされた Web ページを実時間で峻別するアルゴリズムが求められる。本研究では、観測期間に依存しない公正な判別方法を有限の計算機リソースの中で実現するためにアクセス間隔に着目し、長期間に渡ってアクセス履歴を効率的に記録可能なデータ構造を考案した。そして、試作した稀な Web アクセスを検出するシステム (Web-Prospector) を本学学内ネットワークに設置し、提案システムの動作検証を行った。

16 日間の検出実験の結果、観測された URL の総数は約 2,900 万個となった。稀な URL の検出実験では判定条件を固定し、学習期間を変化させて稀にアクセスされた URL の総数を計測した。

提案システムは実時間検出が可能であるが、本方式では膨大な URL 文字列をメインメモリ内に記録することが困難であることが判明した。この問題に対し、ハッシュテーブルを動的に拡張可能にすることで静的に確保される固定領域を削減することや、パトリシアトライの導入によって URL 文字列を圧縮するなど、本問題を解決する手法について考察した。

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究の背景

1991 年に World Wide Web(以下、Web[9] と呼ぶ) が産声を上げてから、約 20 年の内にインターネットのトラフィックの主流はそれまで大半を占めていた FTP から、HTTP を用いた Web アクセスに取って代わられた。Web がインターネットに与えたインパクトの中で他のプロトコルを使ったアプリケーションと大きく異なる点の一つは、それまで主に研究者の間で使われてきたインターネットという新しいメディアの利用者層を一般大衆にまで幅広く広げる契機となったことである。Web を通じてインターネットを利用するユーザ層が拡大し、個人によるコンテンツの発信や電子決済、ニュースの閲覧、Web メールの利用など数多くの便利なサービスが実現されてきた。その結果、膨大な数の Web コンテンツが生み出され、ユーザは様々な目的でアクセスしている。ユーザがアクセスする Web ページのアクセス順位とアクセス頻度との間には、べき乗則に従うことが知られているため、一部の人気が高いページにアクセスが集中することは広く知られている。これまでは、インターネット広告を展開する事業者がアクセス頻度を計測してそれが高いページや用語を探すことは積極的に試みられてきた。しかし、Web ページの大半を占めるアクセス頻度が低いページについて、どのようなページが存在し、どのようなアクセスパターンを持つのか、その詳細を調査する試みは未だになされていない。その理由として、それらを解明することに事業として利益を伴わないことや、アクセス頻度が低い Web ページは膨大な数となるためすべてを把握することは大変困難であることが上げられる。しかし、それらのページやページへのアクセスを明らかにすることは、急成長した Web のユーザとコンテンツの間で形作られるネットワークモデルやアクセスモデルを明らかにする貴重なデータとなる。そのデータの中には、悪意のあるユーザによるマルウェアの感染活動や攻撃が行われたアクセスが含まれている可能性もある。

1.2　研究の目的

稀なアクセスはめったに起こらないため、本研究は長期間に渡って稀にアクセスされるページへのアクセスとそのアクセスパターンをリアルタイムに解明するための観測システムを開発することが目的である。しかし、稀にアクセスされる Web ページは Web のコンテンツ全体の中で膨大な数を占めるため、それを現在の一般的な計算機のリソースの中に記録し、稀なアクセスであるかどうか判別することは極めて困難である。そのため、それを実現する手法を考案し、様々な観測

条件の元でシステムとしてまとめあげる手法を提案することが目的である。実際に観測をの存在を明らかにし、それらへの特異なアクセスパターンを検出することで様々な目的でアクセスを行うユーザ層の活動を解明する糸口をつかむことができる。しかし、アクセス頻度が低い稀な Web ページの数は膨大な数に及ぶため、長期間に渡って稀にアクセスされた Web ページを検出するには、アクセスされた Web ページの URL とアクセス頻度をすべて記録する必要がある。それを一般の計算機のリソースの内に記録し、そこから稀なアクセスを検出することは大変難しい。そのために、一部のアクセス頻度が高い Web ページだけでなく、ほとんどアクセスされることのない膨大な数に上るアクセス頻度が低い Web ページまで検出対象を広げ、それらを観測可能なシステムを実現することで、これまで知られていなかった Web の生態を知る新たな知見を得ることが目的である。

第2章　WWWの変遷とWebアクセスに関連した既存技術について

2.1　World Wide Webの誕生と提供コンテンツの拡大

2.1.1　World Wide Webの誕生

World Wide Web[9] は、1991 年に欧州原子核研究機構において情報システムのコンサルタントとして勤務していた Tim Berners-Lee によってハイパーテキストによる文書管理を目的として創造され、当時彼が使用していたマシンである NextStep 上に世界初の Web Server が実装された。Web が開発される以前では、CERN で働く研究者達は研究論文やソフトウェアのマニュアル、ニュースグループの記事あるいは会議の議事録といった様々な用途の文書を複数の互いに異なるアーキテクチャのマシン、異なる OS、異なる文字コード規格、異なるアプリケーション用フォーマットのもとに保存、閲覧、編集していた。

このように、用途に応じて異なる規格で管理されることが一般的な環境であった中で、Web が提示した次に示す 3 つの規格 (URI[7], HTML, HTTP[6]) によってそれらの差異を吸収し、汎用的な文書管理システムを実現した。

2.2　Webを構成する要素

2.2.1　Uniform Resource Identifier(URI)

URI は、HTML で書かれた文書を指し示す文書や画像などのファイルに限定されず、あらゆる情報を識別する汎用的な概念である。例えば、ハイパーテキスト内で文章として提示した概念そのものを指したり、人物の特徴となる情報を指したりすることもできる。そのため、URI が対象とするものは、あらゆる情報そのものである。URI は、その対象にアクセスする手段やネットワーク上のアドレス、名前などその対象に関するあらゆる属性が含まれる。例えば、その対象へアクセスする手段として、http:スキームや ftp:スキーム、手元で稼働しているマシンのストレージを示す file:スキームなど、スキームと呼ばれるアクセス手段を表す集合や、そのスキームから始まり、対象となる情報のネットワーク上の位置を指し示す URL(Universal Resouce Locator), ネットワーク上のリソースの対象を一意の名前で識別するための URN(Universal Resouce Name) と呼ばれる名前空間が定義されている。

2.2.2 Uniform Resource Locator(URL)

URL[7] は、ネットワーク上のリソースの場所を指し示す指示子である。先頭には、対象までアクセスするための手段を示すスキームが付与される。次に、そのリソースがネットワーク上に存在する場合には、スキームの後ろに"//"が付けられる。そして、リソースが存在するマシンのホスト名、そのホスト上でのリソースへのパスを含む対象となる情報の名前を記述する。http URL の場合では、閲覧中の Web ページに入力事項があったり、クライアントの要求を入れたクエリーと呼ばれる文字列が"?"で区切られた後に続く。

2.2.3 Hyper Text Markup Language(HTML)

HTML は、文書の見出しやフォント、色などの文書構造をタグと呼ばれる命令を用いて定義する言語である。例えば、<h1> </h1> というタグで囲まれた間の文字列は、その文書の見出しとして表示されるようにその文字列の前後に改行後が入り、太字で表示されるように強調される。これらのタグは、クライアントが使用するブラウザによって解釈、整形され、適切な画面を作り上げる。タグの中には、他のハイパーテキストとのリンク構造を埋め込んだアンカータグと呼ばれるものもある。リンクを示すタグとリンク先の URL が書かれており、クリックすることでブラウザはその URL が示す文書を取得するように Web サーバへリクエストを発行し、サーバは適切な文書を返す。

2.2.4 まとめ

上記 3 つの概念および規格によって、Web は異なる複数のノードに分散しているハイパーテキストを URI によって指し示し、Hyper Link と呼ばれるハイパーテキスト間を繋ぐリンクを文書構造の中に定義することでハイパーテキスト間にリンクを形成した。ネットワーク上のリソースを指し示す新たなアドレス体系である URI と Web 上のコンテンツを送受信する専用のプロトコルである HTTP によって、新たなアプリケーションレイヤを構築し、利用者は物理的な距離や規格を意識することなくハイパーテキストによる仮想的な情報空間を直感的に行き来することができるようになった。さらに、この情報空間はネットワークの中心にハイパーテキストそのものやそれを示すメタデータを集めたデータベースなどは必要ないため、非中央集権的に形成される。この点は、新しいインフラとして生まれたインターネットとの親和性が高い。

2.2.5 Web の隆盛

Web が発明された初期では、コンテンツは HTML のみで記述され、あらかじめサーバが保管している静的な HTML ファイルをユーザのリクエストに従ってそのまま送り返すだけであり、一方通行の通信であった。次第にサービスを展開する事業者は、Web 上での売買など様々なサービス

を実現するために、ユーザとの双方向の通信を志向するようになった。例えば、ユーザが望むサービスを Web を通して受け付けたり、取引をする上で必要な情報を入力してもらうためには、ユーザとの対話的な通信が不可欠だからである。そのため、Web サーバは、あらかじめ提示した選択肢の中からユーザがリクエストした内容に応じて、適切な情報を組み込んだコンテンツを動的に生成したり、データベースへ問い合わせてデータの記録、探索を行ったりするなど様々なふるまいを記述する必要が生じた。これを実現するには、文書の構造化記述言語である HTML だけでは機能不足となり、ユーザのリクエストの解釈やそのリクエストごとの条件判断、データベースなどの他のアプリケーションと連携する必要が生じ、Web アプリケーションとその他のアプリケーションを連携させるための枠組みや Web アプリケーションのための高水準プログラミング言語がいくつか開発された。これらの技術は、それを実行する場所に応じて大きく分けて 2 つの形態がある。

1. サーバサイドで動的 Web コンテンツを生成するための枠組みやプログラミング言語
   - (例) CGI(Common Gateway Interface), PHP(PHP:Hypertext Preprocessor)
2. クライアントサイドで動的 Web コンテンツを生成するための枠組みやプログラミング言語
   - (例）JavaScript, Jave Applet, ActiveX Control

2.2.6 各要素技術の概要

CGI CGI は、Web サーバがバックエンドで動いているデータベースなどの他のアプリケーションとデータをやりとりするための共通のアプリケーションインターフェースである。プログラミング言語は任意のものでよく、主に Perl や Python がよく使われている。

PHP PHP は、サーバサイドで動的 Web ページを生成するために用いられるプログラム言語の中で代表的な言語である。PHP は、通常の HTML タグの中に埋め込まれて実行される言語であり、基本的な制御構造の他、文字列の加工からフォームへの入力、Web サーバとデータベースを連携させるための入出力ライブラリなど豊富な機能を有している。

Java Applet Java Applet は、Sun Microsystems(現 Oracle 株式会社) が開発した Java VM 上で実行されるプログラムである。アプレットは HTML ページに埋め込むことができ、JVM を利用可能なブラウザ上で実行される。Java Applet は、クライアントの仮想マシン上で実行されるため理論的にはセキュリティーに強い実行方式だが、現実には Java VM のバグや脆弱性によって必ずしも安全ではないものとなっている。

JavaScript JavaScript は、よりユーザのふるまいを細かく把握する機能が充実したクライアントサイドで実行されるプログラミング言語である。例えば、JavaScipt はブラウザ上でユーザが指し

ているマウスの位置や接触しているか否か、などユーザの動きに応じて様々なイベントを容易に記述することが可能な言語である。

まとめ　以上のような技術によって、利用者は豊かなコンテンツの恩恵を受けることができるようになった。しかし、その自由度の高さ故にそれらの技術に潜む潜在的な脆弱性を悪用した悪意のある攻撃やセキュリティー上のリスクを抱えることとなり、実際に様々な被害に遭う事例が発生している。Web の利用者にとって、近年もっとも深刻な被害を起こす攻撃事例は、次に説明する Drive by Download と呼ばれる手法によるマルウェアの感染である。

2.3　Web を介したマルウェアによる感染活動

Gumblar に代表される Drive by Download 攻撃は、一般のクライアントノードに潜むセキュリティ上の脆弱性を突いてマルウェアをダウンロードさせることで感染させる手法であり、Web サイト閲覧時の大きな脅威となっている。以下に、Dribe by Download 攻撃の概要を述べる。Drive by Download 攻撃は、まず SQL インジェクションや Web サーバの FTP アカウントを盗むことによって悪意のあるユーザが正規 Web ページのコンテンツを改ざんし、その Web ページへアクセスしたクライアントノードを攻撃サイトやそれに繋がる踏み台サイトへ強制的に転送させるスクリプトが埋め込まれる。これには、ウィルス対策ソフトや Web サーバ管理者に気づかれないように難読化された javascript によるリダイレクト命令が埋め込まれていることが多い。

悪意のあるユーザによってコンテンツが改竄された正規 Web ページへアクセスしたクライアントノードは、強制的に踏み台サイトや攻撃サイトへ転送され、そこでクライアントノードにインストールされている OS やブラウザ、あるいはブラウザのプラグインなどの脆弱性を突いてマルウェア本体のプログラムをダウンロードするためのダウンローダが注入される。ダウンローダを注入されたノードが十分な数に達したり、あらかじめセットされた一定の期間が経過すると、マルウェア本体のプログラムのダウンロードが始まり、マルウェアに感染する。

マルウェアに感染するとクライアントノードから ID やパスワードなどの個人情報が漏洩したり、他の感染したマシンと共に悪意のあるユーザによってマシンの制御を奪われ、ボットとなって DDoS(Distributed Denial of Service) 攻撃に利用される事例が発生している。

ここで注目すべきことは、踏み台サイトや攻撃サイトへ転送される際にアクセス先として指定される URL は、一般のユーザがめったにアクセスしない URL であることである。これには、twitter のように入力文字数に制限のあるサービスにおいて本来の URL 文字列を短い文字列に置き換えた短縮 URL が悪用されることもある。この場合、twitter を閲覧しているユーザはリダイレクト先の Web ページが悪意のあるページとは知らずにクリックしてしまい攻撃サイトへ転送されてしまう。

2.4 アクセス検出を行う既存の技術やシステム

2.4.1 IDS

IDS(Intrution Detection System) は、ネットワークやホストに対して流れ込むパケットを検査して悪意のあるユーザからのアクセスを検知して所有者に知らせるシステムである。検知手法としては、パケットのペイロードのデータのパターンからあらかじめデータベースに登録された文字列の並びなどのパターンに一致していないかマッチングを行うことで異常を検知する。その他には、トラフィック量を常時監視して履歴として保存し、既知のトラフィックパターンと大きく異なる急激な変動に対して異常と検知し、利用者に知らせる機能がある。

2.4.2 DPI

DPI は、パケットのペイロードの中まで深く検査し、そこに書かれた検索ワードやリクエスト先の URL などを抽出してユーザの嗜好やアクセス履歴を調査したり、コンピュータウィルスによる侵入やスパムメールなどを識別してそのパケットをブロックしたり、転送するなどより高機能なサービスを実現している。

2.5 Web アクセスにおけるアクセス頻度と宛先 Web サーバの関係

Web ページは、世界中に膨大な量が存在するが、それらに対するアクセス頻度と、それをもとにした順位 (Rank) との間には、一定の法則に従うことが知られている。両対数軸グラフの横軸にアクセス先の Web ページをアクセス順位ごとに並べ、その順位に対応したのアクセス頻度を縦軸にプロットすると、ほぼ直線となる。これは、Web ページの順位とアクセス頻度の関係がべき乗則に従うことを示している。べき乗則に従う例は、地震の規模と開放されるエネルギーや本の中に出現する単語の出現頻度とその順位の関係を表すジップ分布、収入とその人数の分布を表すパレートの法則など事象に対しても見られる法則である。一般的にべき乗則を数式で表すと、次のようになる。[8]

$$p(k) = Nk^{-r} \tag{2.1}$$

変数 k はネットワークの次数を表す。$p(k)$ は、次数 k を持つ Web サーバが全 Web サーバ N に対して出現する確率的な割合を示す。N は、確率分布において式 2.2 を満たす規格化定数であり、式 2.3 によって求められる。

$$1 = \int_{k_{min}}^{\infty} p(k)dk = \frac{N}{1-\gamma}[k^{-(\gamma+1)}] \tag{2.2}$$

$$N = (\gamma - 1)k_{min}^{\gamma-1} \tag{2.3}$$

2.6　本研究において注目する観測領域

図 2.1 のように、Web ページのアクセス頻度とその順位の間には、べき乗則が成り立つことが知られており [5]、アクセス頻度が高い (人気の高い)Web ページは全体の中でごく僅かであることが知られている。それに対して、アクセス頻度が低い (人気の低い) ページは膨大な数となる。アクセス頻度が高い Web ページを検出する場合、大半のアクセス頻度が低い Web ページについては詳細にそのアクセスを記録する必要はなく、一部のアクセス頻度が高いページに限定して記録すればよい。しかし、本研究は稀にアクセスされる Web ページを検出するために、アクセス頻度が低いページに着目するため、図 2.2 の領域 B の Web ページ群へのアクセスに注目する。図 2.2 から見て取れるように、アクセスされることが稀な Web ページはインターネット上の Web ページの大半を占めるため、稀にアクセスされる Web ページを検出するためにはインターネット上のほぼすべての Web ページへのアクセスを扱わなければならない。

図 2.1: べき乗則に従うグラフの例

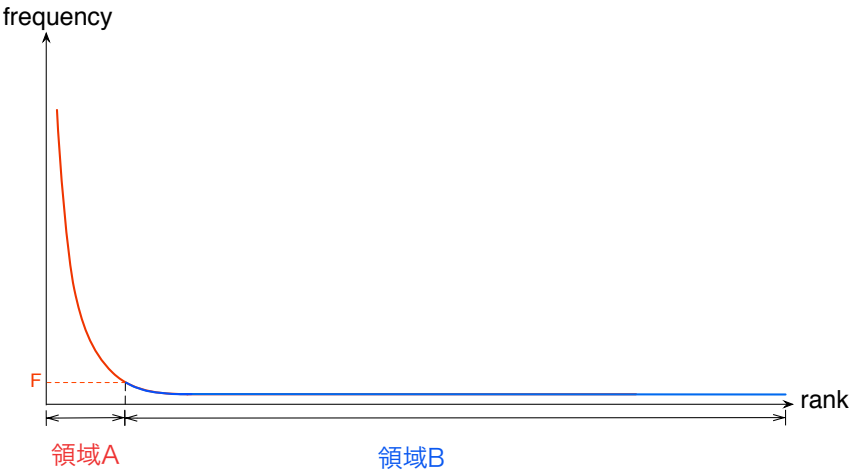

図 2.2: 本研究で注目する観測領域 B

2.7 実トラフィックにみる Web サーバのアクセス頻度と順位の関係

アクセス頻度の観点から Web ページの分布を調べることは、稀にアクセスされる Web ページの総量を把握する上で重要なファクターとなり、本研究の目標である稀にアクセスされる Web ページを検出するシステムを設計する上で考慮すべき重要な要素となる。そこで、検出システムを設計する前に、現在のインターネット上においてもアクセス頻度の観点で捉えた場合にべき乗則が成り立つか検証した。検証作業に用いた実トラフィックデータは、WIDE Project で日本国内のバックボーンネットワークのトラフィックを計測している MAWI Working Group[2] から提供して頂いたデータである。このデータは、複数の太平洋横断ケーブルの一つを観測している samplepoint-F において 2006 年 6 月 24 日 14 : 00 ~ 14 : 15 までの 15 分間のデータである。このデータから宛先ポート番号が 80 番、443 番、8080 番へのアクセス回数を計測した結果を、図 2.3 に示す。縦軸がアクセス頻度の累積値、横軸がそれに基づくアクセス順位である。また、図 2.4 は、両軸を対数にして表示したグラフである。図 2.4 の両対数グラフにおいて、ほぼ直線を示していることから現在のインターネット上でも Web サーバのアクセス頻度とそれに基づく順位にはべき乗則が成り立つことが改めて確認された。

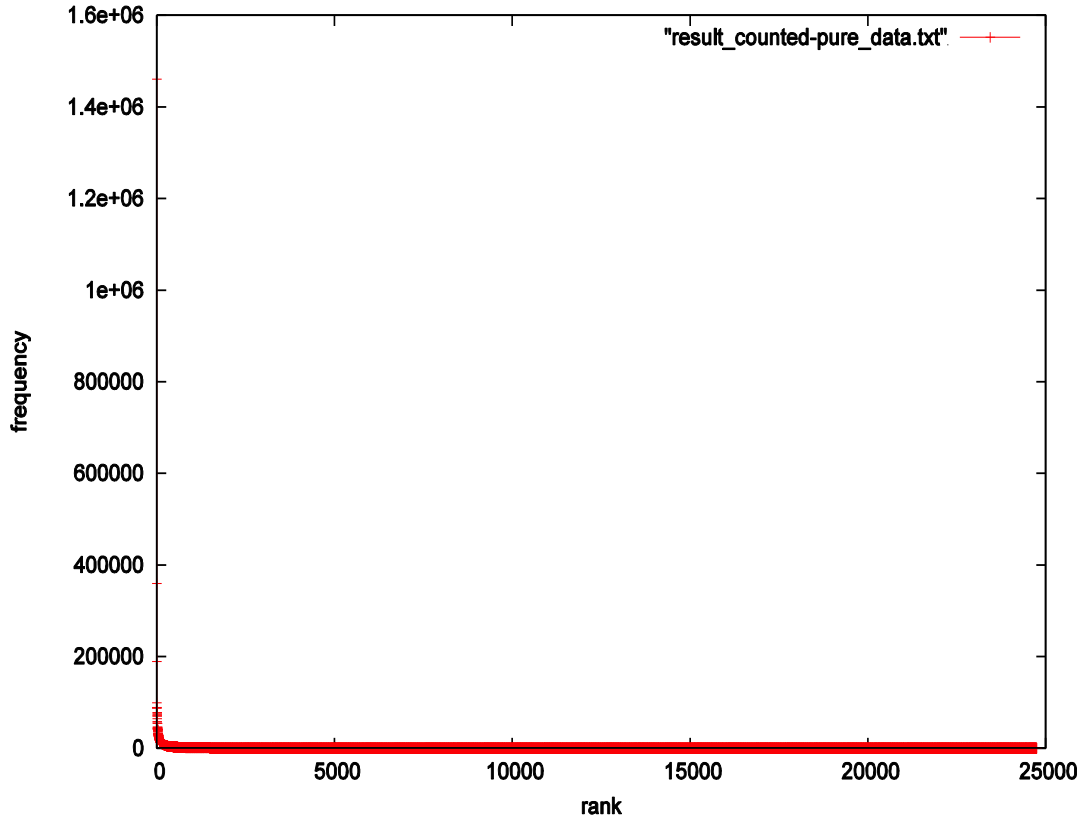

図 2.3: 実トラフィックデータから得られたアクセス頻度とそれに基づく順位の関係

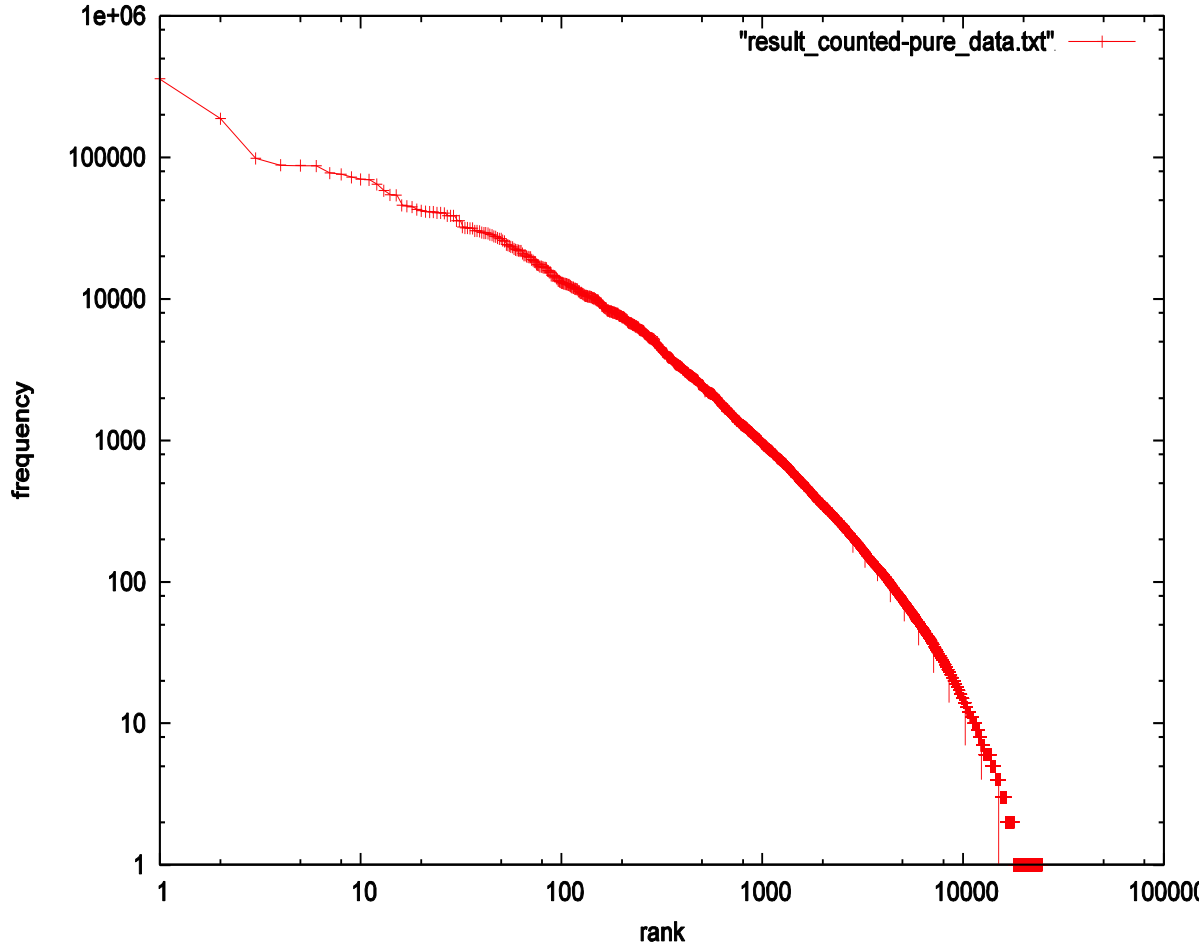

図 2.4: 実トラフィックデータから得られたアクセス頻度とそれに基づく順位の関係 (両対数)

第3章　稀なWebページへのアクセスを検出するシステム:Web-Prospectorの提案

3.1　提案するシステム

インターネット上に存在するWebページをページ数で比較した場合、アクセス頻度が低いWebページは大量に存在することが分かる。しかし、これまではインターネット広告業者や検索サイト等で注目されてきたWebページは、専ら少数のアクセス頻度が高い(人気の高い)Webページに絞られていた。これに対して、アクセス頻度が低いWebページへのアクセスには目立った特徴や有用な情報が存在していない印象を受ける。しかし、アクセス頻度という観点ではなくアクセス目的で比較した場合、事情は大きく変わってくる。World Wide Webは、その発祥から今日に至るまで爆発的なスピードで普及した結果、様々な国籍や年齢層、職業を持った人達によって利用されている。それによって、Webを利用する目的は極めて多様化した。そのため、たとえアクセス頻度が低いURLであっても、アクセス頻度が低いWebページが膨大に存在するため、稀なWebページへのアクセスは実に多様なアクセス目的を内包しているといえる。そこには、通常のWebアクセスだけでなく例えばDrive by Download手法を用いたマルウェアによって、攻撃ページへ誘導された際のアクセスも含まれている可能性もあり、稀にアクセスされるWebページの数だけアクセス目的が存在すると言っても過言ではない。そこで、本研究ではこれまで本格的に注目されることがなかったアクセス頻度が低いWebページ群に焦点を当て、それを検出するシステムの提案を行う。検出システムを実現するには、稀にアクセスされるWebページを検出するために出現頻度の高低に関わらずアクセスされたあらゆるWebページへの履歴を記憶しておく必要がある。そのために、まずWebページをURLによって識別し、膨大な数となるアクセス頻度の低いWebページのURLを効率的に記録する必要がある。次に、そのURLへのアクセス履歴をもとにアクセスの「稀さ」を算出し、それを満たすURLをオペレータに通知する機能が必要である。これらの機能を持ったシステムは、セキュリティ上の観点から1組織(AS)にごとに1台設置され、内部の利用者のアクセス履歴が漏洩することがないように十分に配慮された運用形態を採るべきである。したがって、現在クラウドコンピューティングとして注目されている大量のPCノードを接続した大規模分散システムではなく、平均的なPCサーバが持つ計算機資源内で実現可能なシステムであることが望ましい。そして、このシステムを一般的なPCサーバが持つ計算機資源の枠内で実現することは学術的に大きな挑戦といえる。また、稀にアクセスされるWebページへのアクセスやその傾向が、インターネット全体のトラフィックに与える影響について新たな知見を得る貴重な

判断材料になることも期待される。

稀な Web ページへのアクセスを検出するためには、必然的に長期間に渡ってあらゆる Web ページへのアクセス履歴を把握しておく必要がある。しかし、それによって膨大な記録データが生じるため、それを有限のハードウェアリソース内に収めるための容量効率が高い記録データ構造やアルゴリズム、そして少ないデータ量で稀さを決定可能な効率的かつ有効なパラメータの選別が重要である。そこで、まず本研究が対象とする「稀なアクセス」の定義と、それを元に提案システムに求められる機能について具体的に述べていく。

3.2 本研究が対象とする「稀なアクセス」

冒頭で述べたように現在の World Wide Web 上には、ホームページの閲覧だけでなく Web-Mail や http を用いたファイル転送サービス、電子決済、ニュース、地図情報など多様なサービスが展開されているため、多様なアクセス目的のもとに様々なアクセスパターンが存在している。例えば、天気情報のように定期的に毎日アクセスされるページや、Web ページを巡回するクローラーのように数週間から数ヶ月単位で定期的にアクセスされるページもある。逆に、趣味のページのように人によって不定期にアクセスされるページも含まれている。

定期的にアクセスされるページに対して、ある時突然アクセスが途絶えたり、定常的な間隔から大きくかけ離れたギャップを生じてアクセスされた場合、そのアクセスはそれ以前のパターンに対して余り見られないアクセスである。つまり「稀なアクセス」といえる。

不定期なアクセスに対しては、例えば前回アクセス時刻から一年を経てアクセスされた場合、その期間がユーザにとって「稀」と感じる長さであれば、それは「稀にアクセスされた」といえる。

以上を踏まえ、本研究はアクセスの「稀さ」の指標として、アクセス間隔に着目する。しかし、稀であると判定する普遍的な「長さ」は存在せず、それはユーザの趣向や感覚によって大きく変化する主観的な概念である。そのため、それはサービスを提供する人間が目的に応じて最適な長さを決定することが妥当である。本研究において目標とするシステムは、本システムのサービス利用者が稀であると判定した「長さ」(アクセス間隔) を元に、それを満たす URL を有限の計算機資源の中で正確に検出可能なシステムを実現することである。そのため、本システムを特定のユーザに限定して運用するのではなく、多様な価値観を持ったユーザによって汎用的に利用されるシステムであるべきである。

「稀さ」を判定する指標となるアクセス間隔に基づき、様々なアクセスパターンの中から本研究では下記の 2 つのアクセスを「稀なアクセス」と定義した。

1. 観測データ中に存在しない新規の URL へのアクセス
2. (アクセス頻度が α 未満の時) 観測者が定める一定のアクセス間隔 T を越えたアクセス

条件 2. において、アクセス頻度を F 未満とする条件を付加した理由は、アクセス間隔だけに限定した場合、長期観測によって重複した稀な URL を複数回検出することを防止するためである。本定義を元に、アクセス間隔とアクセス頻度を軸にして頻繁にアクセスされる Web ページと稀にアクセスされる Web ページを領域別に図示したものが、図 3.1 である。この図では、左上の赤い色で塗りつぶされた領域が、頻繁にアクセスされる Web ページの領域であり、前述の図 2.2 における領域 A の Web ページ群に相当する。右下の青色で塗りつぶされた領域は、アクセス頻度がサービス利用者が定めた閾値 F より低く、かつ、アクセス間隔が常に閾値 K 以上の長さでアクセスされた Web ページであり、これを稀にアクセスされる Web ページであると判定する。この領域は、前述の図 2.2 では、領域 B の Web ページ群に相当する。

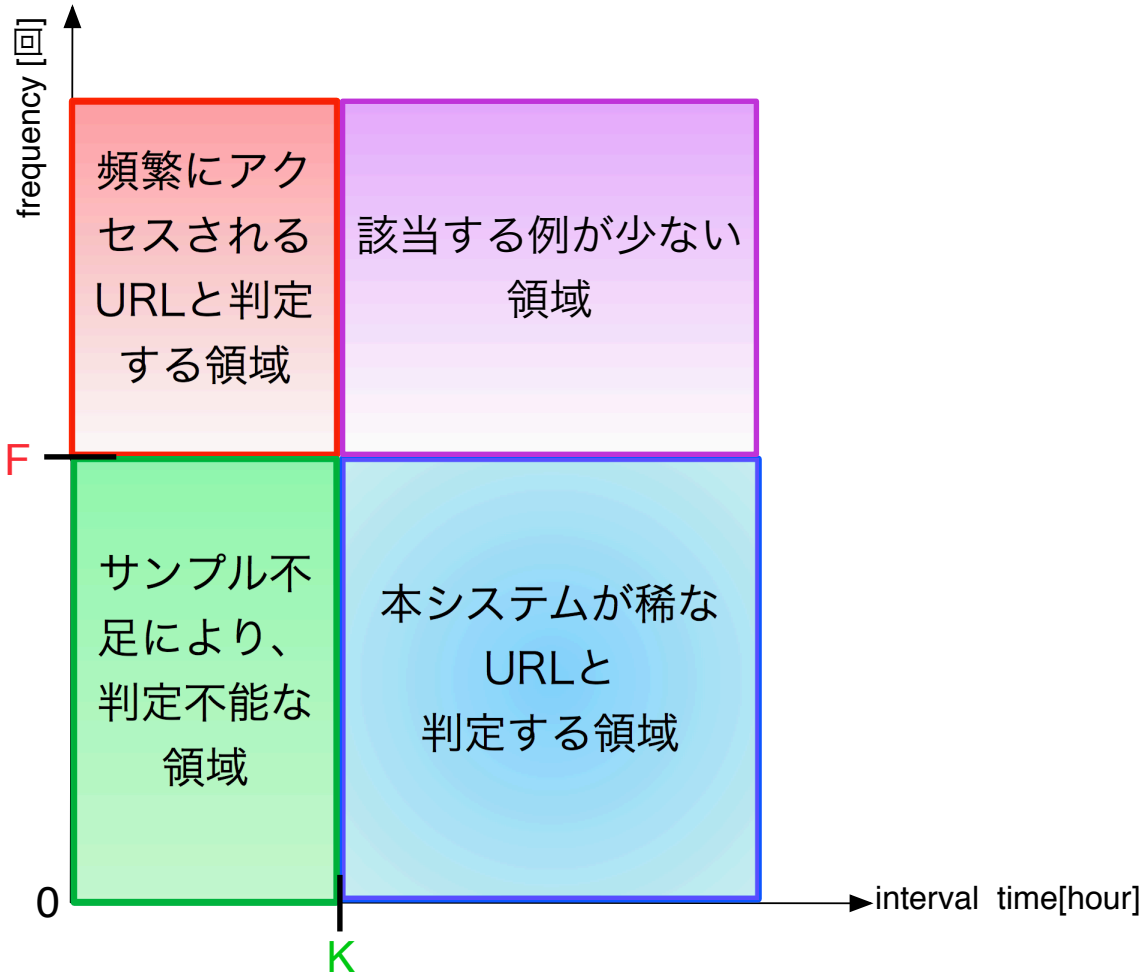

図 3.1: 「アクセス間隔-アクセス頻度」グラフ上で位置づけられる 2 領域

以上に基づき、次節では上記で定義した「稀な Web ページへのアクセス」を検出するために求められる要件について述べていく。

3.3 提案システムの要件

1. 長期観測 (最低 1 年以上) によって生じる膨大な観測データの取り扱いが可能であること
2. 実時間計測及び検出が可能であること
3. 計測期間に依存しない公正な検出手法を有していること

条件 1. が必要な理由は、2 章で述べたベキ分布の関係よりアクセス頻度が高い人気のある Web ページより、アクセス頻度が低い人気のないページの方が圧倒的多数を占めるために、必然的にそれらを検出するためには膨大な数の Web ページの存在を知る必要があるからである。Web ページを識別する指標には、ネットワーク上のリソースの位置を一意に識別する URL がその候補として挙げられる。URL を Web ページの識別子として用いた場合、URL 文字列は数十文字から数千文字に及ぶため、稀にアクセスされたすべての Web ページの URL 文字列のデータ量は膨大なものとなる。さらに、稀にアクセスされる Web ページであるか否か判別するためには、アクセス頻度の累計やアクセス時刻など様々な指標が必要となるため、これらの膨大なデータを有限の計算リソースに落とし込むデータ構造やアルゴリズムが求められる。

条件 2. が必要な理由は、膨大なアクセスデータが発生することが要因の一つにある。提案システムは、長期間の膨大な量のアクセスデータを扱うためにバッチ処理のような一括処理する手法では、アクセスデータを保存するリソースが大量に必要となるため、現実的な手法ではない。その他には、将来的に提案システムを活用していくためには、変化の激しいインターネットの世界で半年前や一年前の検出結果では古いデータとなってしまい、検出データを有効に活かすことができなくなると考えられるからである。そのため、最新の検出結果に基づくアクセス情報を可能な限り迅速に入手可能なシステムにするために、リアルタイムにトラフィックを計測し、その場で検出結果を得ることが可能なシステムが望ましい。

条件 3. の検出手法の問題は、稀な Web アクセスの検出に特有の課題といえる。人気が高い Web ページを検出する需要に応えるためには、例えば現在の流行よりも 10 年前の流行を知りたいと思う人の方が稀であることから、時系列的には計測範囲はより短期的な傾向を持っているといえる。しかし、短期的にはアクセスされることが稀な Web ページを検出するためには、必然的に長期に渡る計測期間が必要でなる。長期間の計測において、アクセス頻度のみで「稀さ」を判別する場合には、アクセス頻度は時系列的に一律に増加し続けるパラメータであるため、計測開始前にしきい値を決定する際に想定した期間が過ぎた場合には、再度適切なしきい値を設定続けなければならない。アクセス頻度が時系列的に線形に増加すると判明している場合には大きな問題にならないが、そのような保証はない。そのため、稀さを判別するしきい値としてアクセス頻度のような時間に依存するパラメータだけを判定基準に採用することは、提案システムを実現する上では不適切である。

3.4 実現方式

提案システムを実現するには、大きく分けて二つのアプローチがある。

1. Web ページに書かれたリンクの URL を辿り続けることで新たな Web ページを検出する
2. パケットキャプチャによって Web ページへのアクセスを検出する

1. では、提案システムがシステム管理者によって与えられた起点となる URL をもとに Web アクセスを開始し、アクセス先の Web ページに存在するすべてのリンクを能動的に辿ることで新しい Web ページを検出する。2. は、ある組織のイントラネットとインターネットの境界で Web ページへのリクエストパケットをキャプチャし、イントラネット内のユーザがアクセスしようとしている Web ページの URL を受動的に取得する。2. の方式の場合、全ユーザのリクエストパケットをキャプチャするため、プライバシの問題や、パスワードの漏洩などセキュリティー上の問題が少なからず存在する。しかし、提案システムがネットワークを管理する組織のオペレータの元で厳格に運用されるならば、それらの問題を回避することは可能である。

提案システムは、2. の方式を採用する。その最も大きな理由として、1. の方式ではリンクが張られていない Web ページを検出することができないからである。稀にアクセスされる Web ページは、リンクが張られているような広く知られた場所に存在することはむしろ稀であると考えられる。

第4章　URL Based Web-Prospectorの設計と実装

4.1　検出システムの構成

URL Based Web-Prospector は、主にパケットキャプチャを行うフロントエンドプロセスと URL 文字列の記録や算出したアクセス間隔とその頻度情報をもとに稀な URL へのアクセスを検出するバックエンドプロセスで構成される。2 つのプロセスに分けた理由は、理想的にはこの 2 つのプロセスは別のハードウェア上で実行されることが望ましいからである。その理由として、フロントエンドプロセスが行うパケットキャプチャ処理は NIC からの割り込み処理が多発するため、パケットドロップを防ぐためには常時計算リソースを十分に確保する必要があるからである。以下に、それぞれのプロセスの詳細な処理内容について述べる。

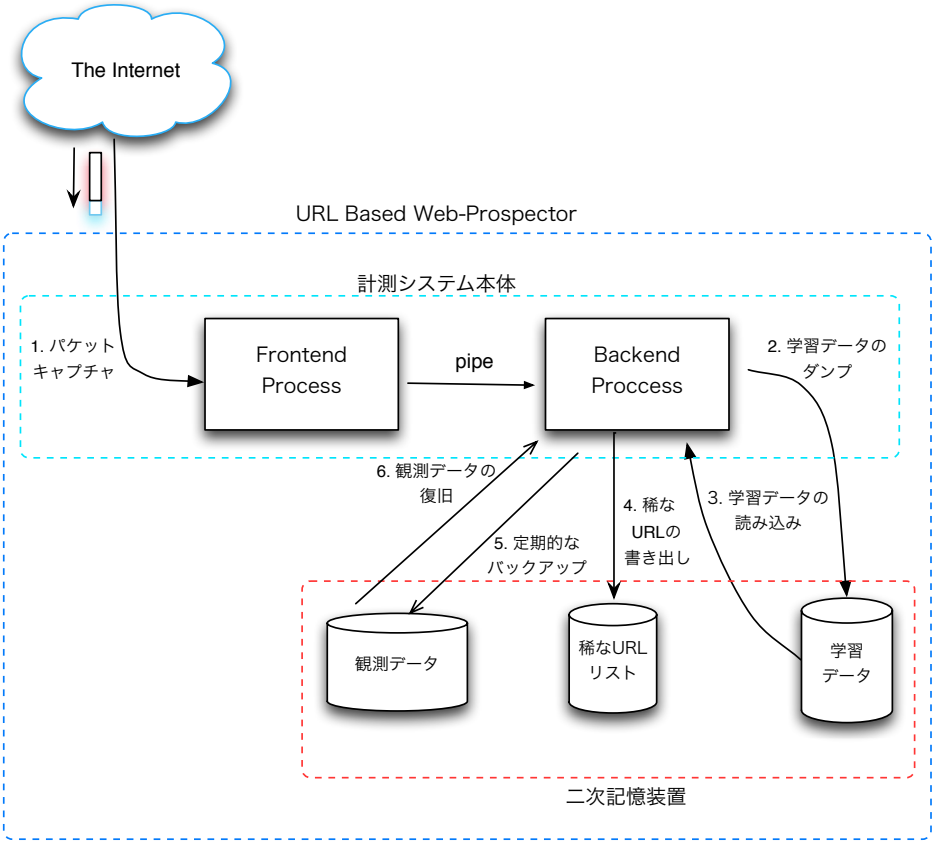

図 4.1: Web-Prospector のシステム構成

4.1.1 フロントエンドプロセス

フロントエンドプロセスは、ネットワーク上を流れる IP パケットの中から Libpcap[1] を用いて、TCP の 80 番ポート宛のパケットだけをフィルタリングして取り込む。次に、取り込まれたパケットのペイロードから図 4.2 のフォーマットに従う"GET /"で始まる Web サーバへのリクエストメッセージの開始を示すキーワードが格納されているか探索し、もし格納されていれば、その Web サーバのホスト名とパス名を取り出す。具体的には、"GET /"で始まり、最初のスペース、またはクエリーの始まりを示す"?"、またはデリゲーションの始まりを示す “#” までをパス名として切り出す。次に、パス名の終わりから 2 つ目のスペースを探し、"Host: "で始まるホスト名の開始を示すキーワードを探す。そこから"\r"または"\r\n"までの文字列をホスト名として切り出す。得られた二つの文字列から、ホスト名の後ろに"/"を付加し、さらにその後ろにパス名を付加して URL 文字列を生成する。次に、アクセス時刻は、取り込まれた pcap 形式のパケットヘッダに書かれているキャプチャ時刻を参照する。最後に、時刻情報と URL 文字列から図 4.3 のようなフォーマットの文字列を生成し、プロセス間通信の一種である名前付きパイプを用いてバックエンドプロセスへ送信する。

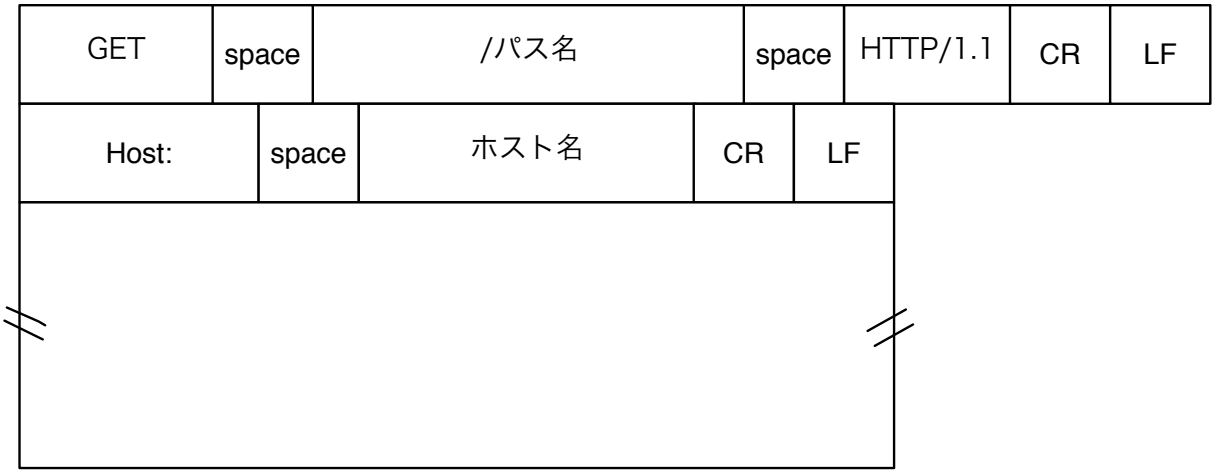

図 4.2: Web サーバへのリクエストメッセージのフォーマット

図 4.3: フロントエンドがバックエンドへ送信するメッセージのフォーマット

図 4.4: フロントエンドがバックエンドへ送信するメッセージの例

4.1.2 バックエンドプロセス

バックエンドプロセスの機能は、主に 5 つある。

1. フロントエンドプロセスが送信した URL 文字列とアクセス時刻が記されたメッセージを受信する。
2. 受信した最新のアクセス時刻からアクセス間隔を求め、その出現頻度を記録する。
3. サービス利用者が設定した稀な URL の判定条件をもとに判定を行い、検出された場合には二次記憶装置に書き出す。
4. システム稼働直後は、学習期間として記録した観測データを学習データとして扱い、学習終了後に二次記憶装置へ書き出し、バックアップを取る。
5. 二次記憶装置へ観測データの定期的なバックアップ。

1. のフロントエンドプロセスがキャプチャした URL とアクセス時刻を収めたメッセージは、プロセス間通信の一つである名前付きパイプによって受信される。

次に、2. のアクセス時刻の差分を記録するためのデータ構造を説明する。バックエンドプロセスの記録データ構造を図 4.5 に示す。

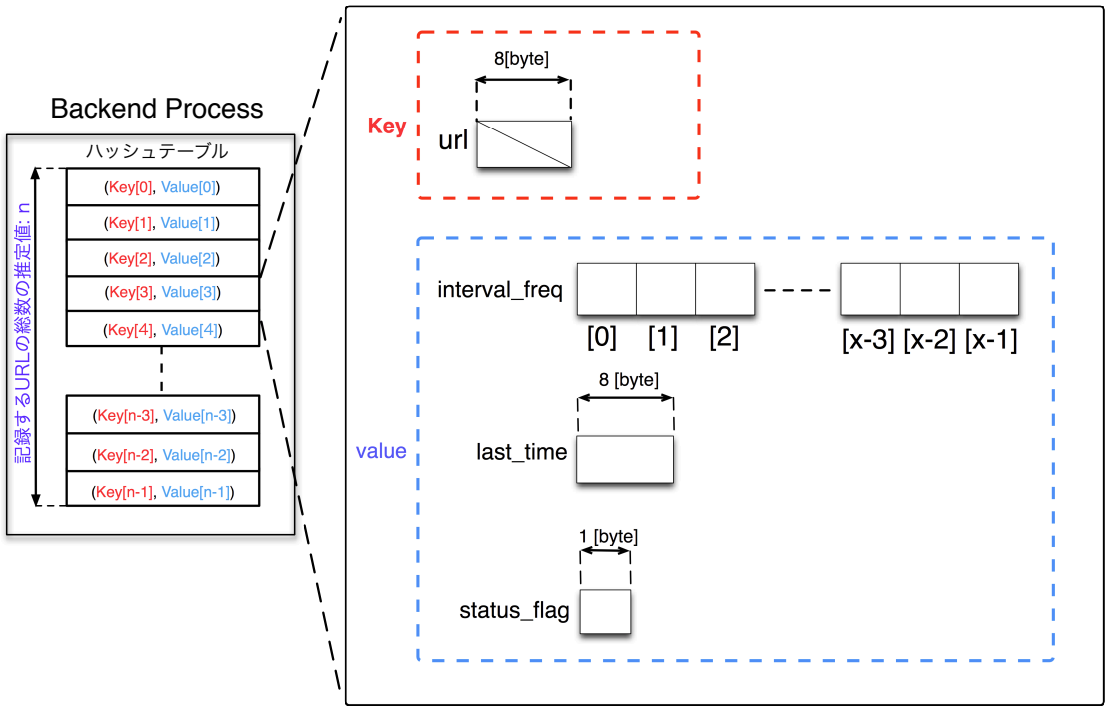

図 4.5: 配列の一要素あたりのデータ構造

変数 url は、URL 文字列を格納しているアドレスを指し示すポインタ変数である。URL 文字列を格納する領域は、malloc 関数によって各 URL 文字列の長さに合わせて動的に記憶領域が確保される。次に、last_time はその URL の最新のアクセス時刻を格納する time_t 型の変数である。

配列 interval_freq は、最新のアクセス時刻から直近のアクセス時刻の差分であるアクセス間隔を算出し、それを 2 の累乗ごとの範囲でに区切ってランク付けして同一ランクのアクセス間隔の頻度を記録する配列である。このアクセス間隔の範囲 (配列の一要素が受け持つ時間の幅) は、指数関数的に増加するため、4.6 のように膨大な時間 (アクセス間隔) を最小限のデータ量で記録することができる。

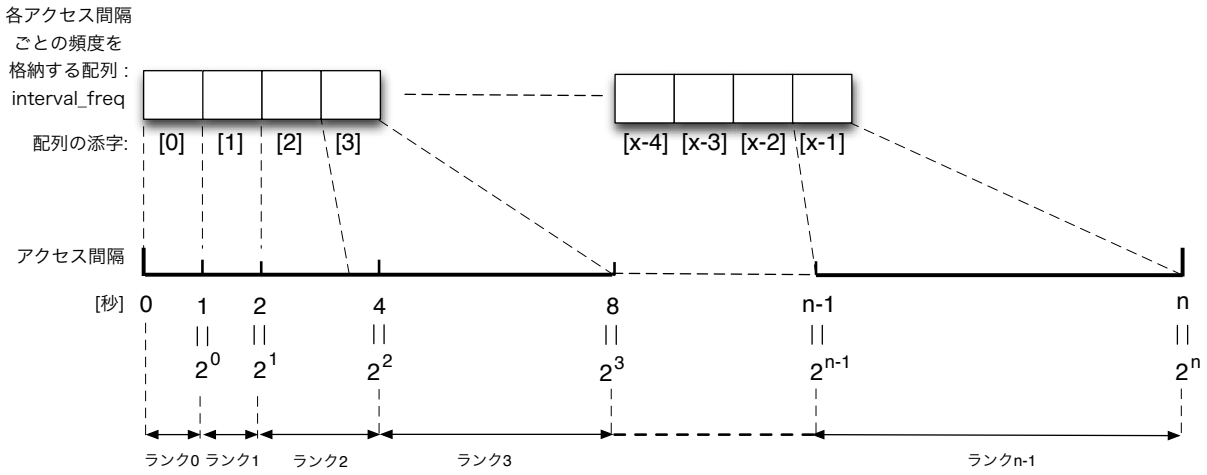

図 4.6: アクセス間隔を記録する配列:interval_freq

以下に、Web アクセスが発生してから該当するランクを指す要素の添字を決定するまでの過程を式にまとめる。ここで、最新のアクセス時刻を表す変数を t_c、直近のアクセス時刻を表す変数を t_l、アクセス間隔を Δt と置く。このとき、該当する配列の要素の添字を x は、式 4.1 と式 4.2 から求められる。

$$\Delta t = t_c - t_l \tag{4.1}$$

$$x = \log_2 \Delta t \tag{4.2}$$

配列 interval_freq の各要素のサイズについては、特に定められていない。観測環境で予想される URL の出現数や計測ハードウェアのメモリ搭載量に応じて最適なサイズを決定するべきである。

もし、メモリ容量が足りないため配列をコンパクトにする必要がある場合には、アクセス間隔が小さい時間帯を間引く方法がある。例えば、アクセス間隔の最大値を 1 年 (= 31,536,000[秒] ≒ 2^{25}[秒]) としたいが、配列を最大 10[個] までしか用意できない場合には、それらの差分である 2^{15}[秒] のアクセス間隔が稀さのしきい値よりも短いならば、その (2^{15}[秒]) 以下のアクセス間隔をはじめから

記録対象から除外すればよい。ここで、間引く量を 2^z[秒] とすると、該当するランクを指す要素の添字は、式 4.3、式 4.4、式 4.5 によって求められる。

$$\Delta t = t_c - t_l \tag{4.3}$$

$$\Delta t' = \frac{\Delta t}{2^z} \tag{4.4}$$

$$x = \log_2 \Delta t' \tag{4.5}$$

キャプチャした URL から適切な配列の要素を参照するには、ハッシュ関数を用いた探索方式の一つであるオープンアドレス法を用いる。

オープンアドレス法

オープンアドレス法は、一つの巨大な配列であるハッシュテーブルに (key,value) ペアを格納し、key を元にその key が格納されている要素 (バケット) を探索するアルゴリズムの一つである。提案システムでは、key は URL 文字列となり、その URL 文字列をもとに一意の数値を出力するハッシュ関数に掛け、得られたハッシュ値が該当するバケットの添字となる。もし、その添字のバケットを参照した際に、既に別の (key,value) ペアによって格納されていた場合、再度ハッシュ関数を実行して新しい添字を得る。この操作を再ハッシュと呼ぶ。空いている要素か、又は削除済みの要素が存在しない限り再ハッシュを繰り返すので、オープンアドレス法ではあらかじめハッシュテーブルの大きさが格納予定の key の総数の 1.1 ~ 1.2 倍程度になるように設計しなければならない。

指定した key の探索処理では、該当する key が見つからなかった場合に再ハッシュを繰り返し、空の要素にたどり着くまで探索を続ける。探索途中に指定された key が見つからず、空の要素にたどり着いた場合には探索失敗となる。

データを削除するには、該当するバケットに削除フラグを付ける。削除フラグの付け方には、key に他のすべての文字コード値と衝突しない特別な数値を設定したり、別途削除フラグとして専用の変数を設けたりする方法がある。削除フラグを付加しない場合、削除されて空となった要素の後ろに探索中の (key,value) ペアが存在していた場合、空の要素に出会った時点で探索処理が終了してしまうため、指定された要素を発見できずに探索処理を失敗してしまうケースが発生する。そのため、削除フラグを設けることで、削除フラグが有効なバケットに出会った場合には、再ハッシュを継続する。

4.1.3 ハッシュ関数の検証

記録アルゴリズムの処理能力を左右する重要な要素であるハッシュ関数は、Web-Proxy サーバのソフトウェアとして広く使われている Squid[3] のソースから/lib/hash.c にある hash_string 関数と hash4 関数を参照した。また、これらと比較用に参考文献 [10] に掲載されているハッシュ関数も参照した。この関数をここでは便宜的に hash_sedgewick 関数と呼ぶことにする。3 つのハッシュ関数に対して, URL 文字列を 100,584, 1,167,087, 10,276,323[個] 入力させた時の衝突回数 (frequency) と同一の衝突回数を示したその累計値 (frequency of frequency) のグラフを図 4.7~ 図 4.15 に示した。

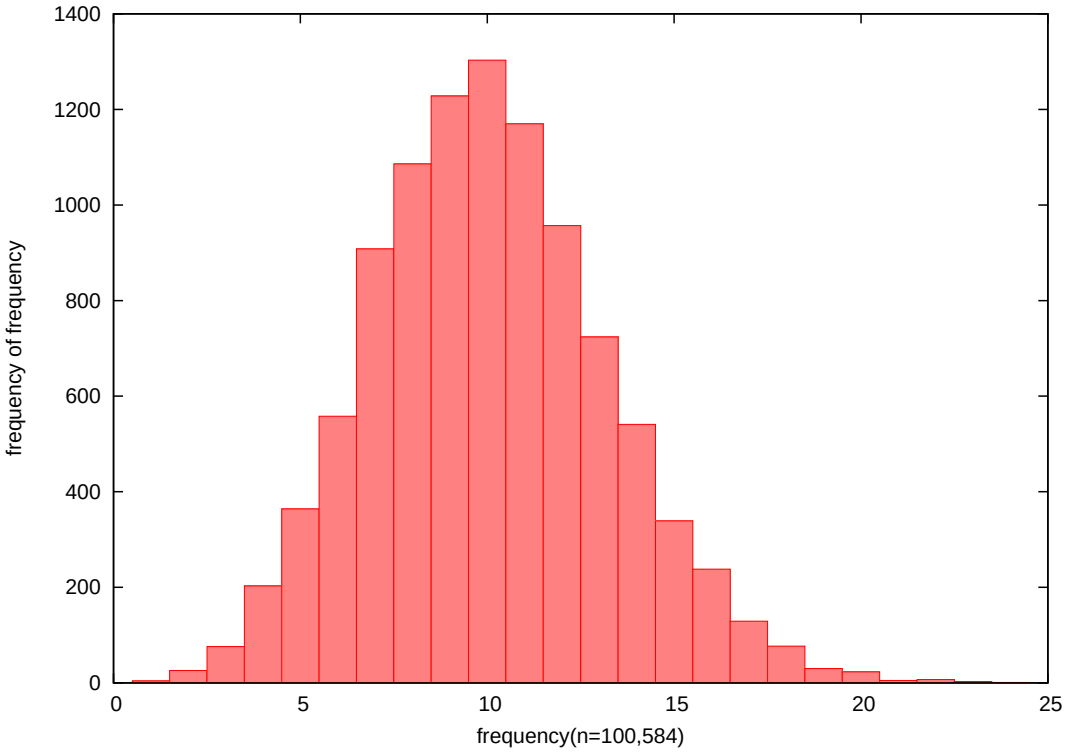

図 4.7: hash4 関数の衝突耐性, 入力 URL 数：100,584[個]

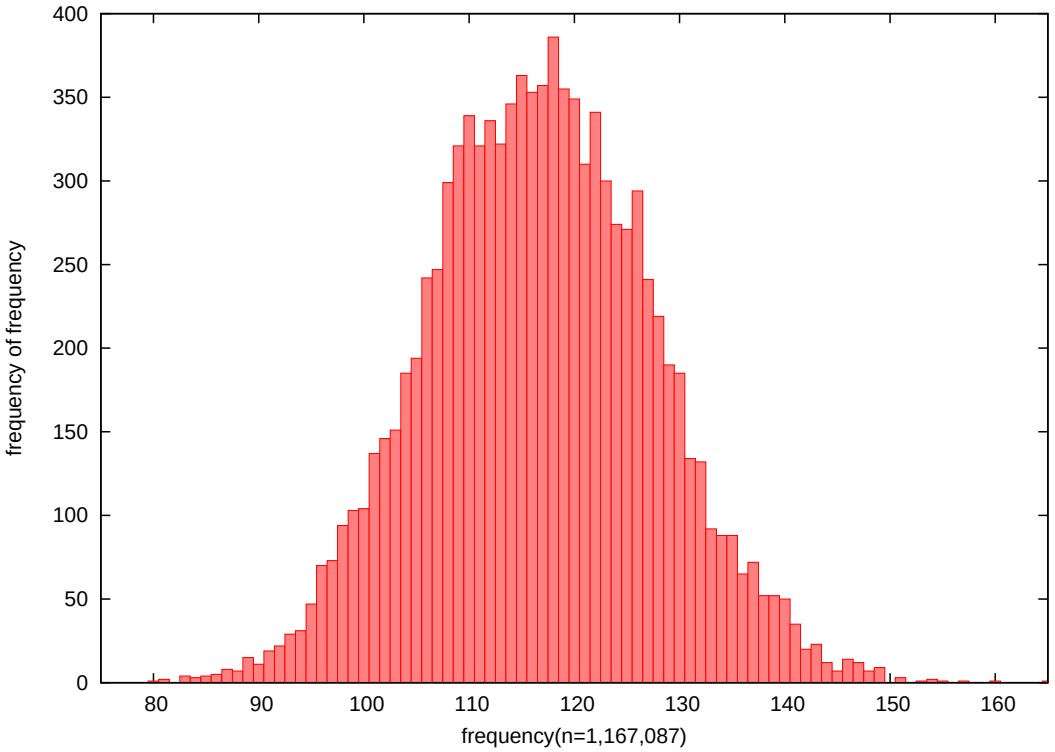

図 4.8: hash4 関数の衝突耐性, 入力 URL 数：1,167,087[個]

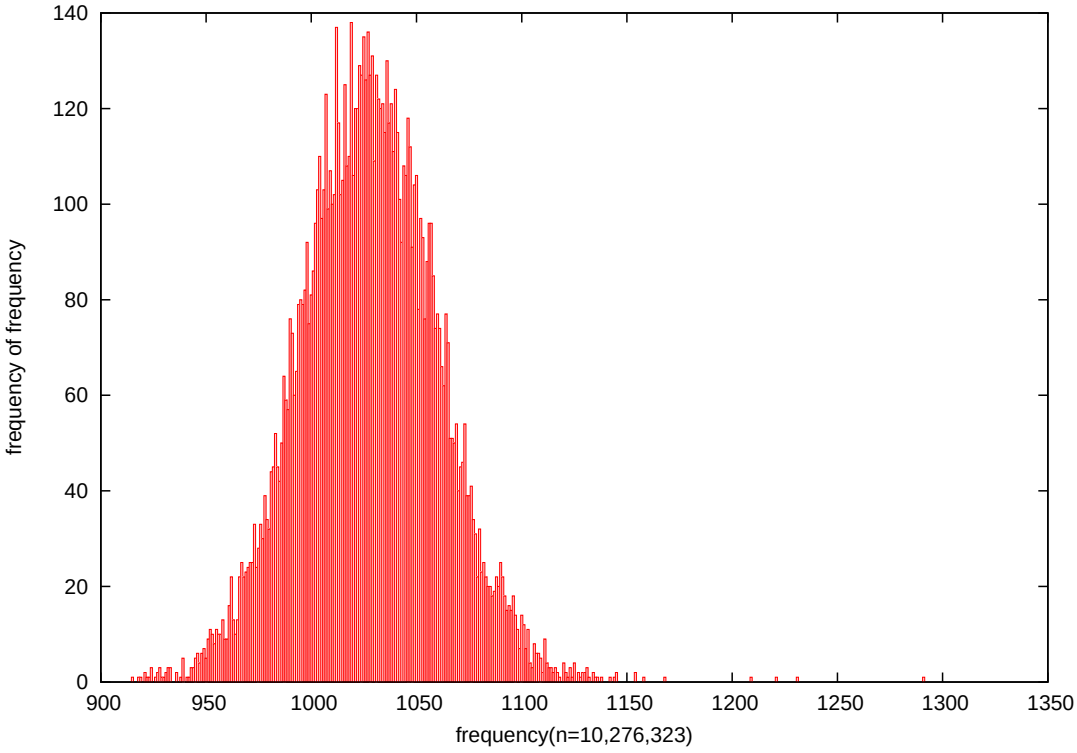

図 4.9: hash4 関数の衝突耐性, 入力 URL 数：10,276,323[個]

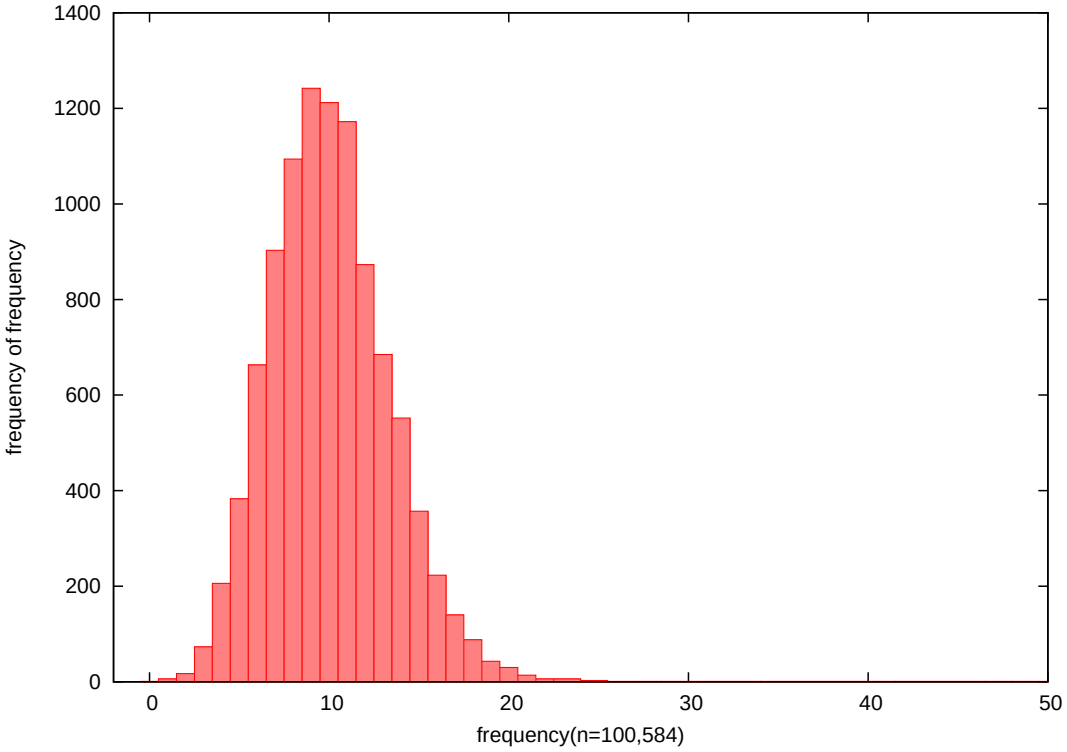

図 4.10: hash_string 関数の衝突耐性, 入力 URL 数:100,584[個]

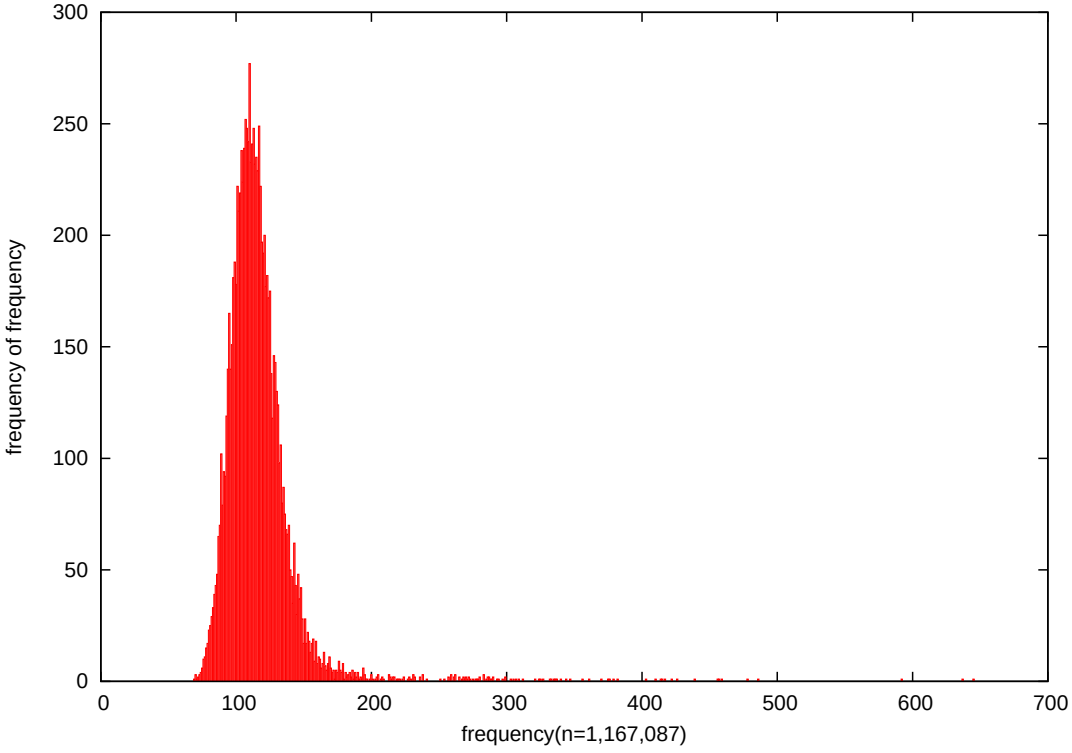

図 4.11: hash_string 関数の衝突耐性, 入力 URL 数:1,167,087[個]

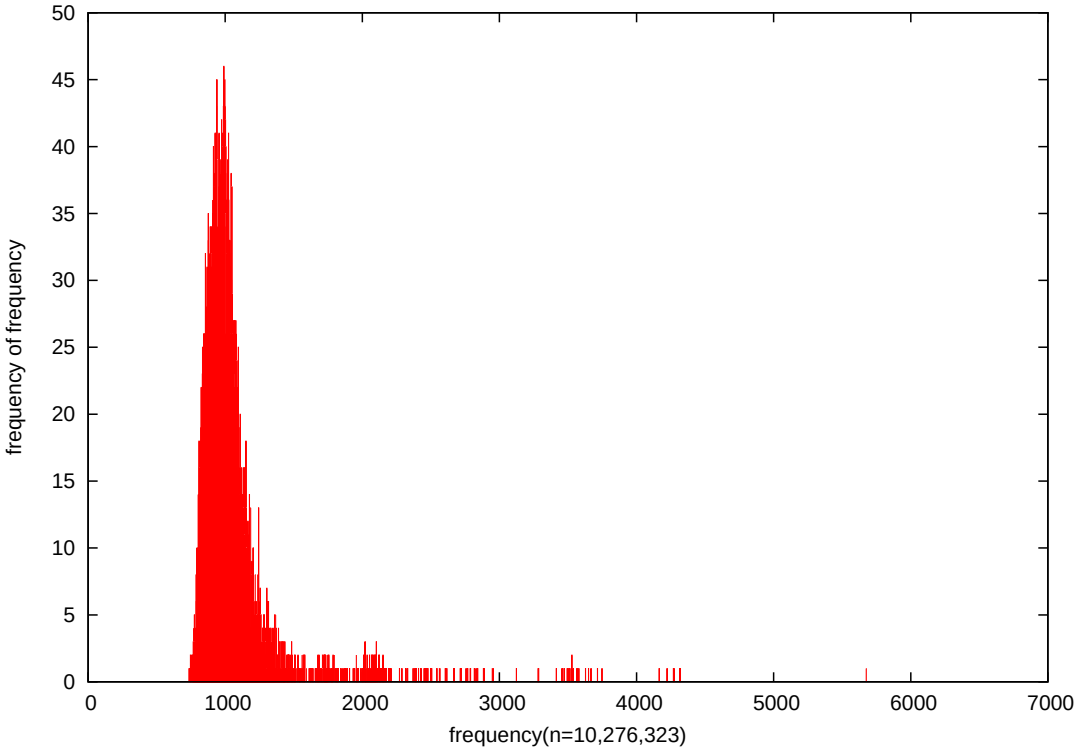

図 4.12: hash_string 関数の衝突耐性, 入力 URL 数:10,276,323[個]

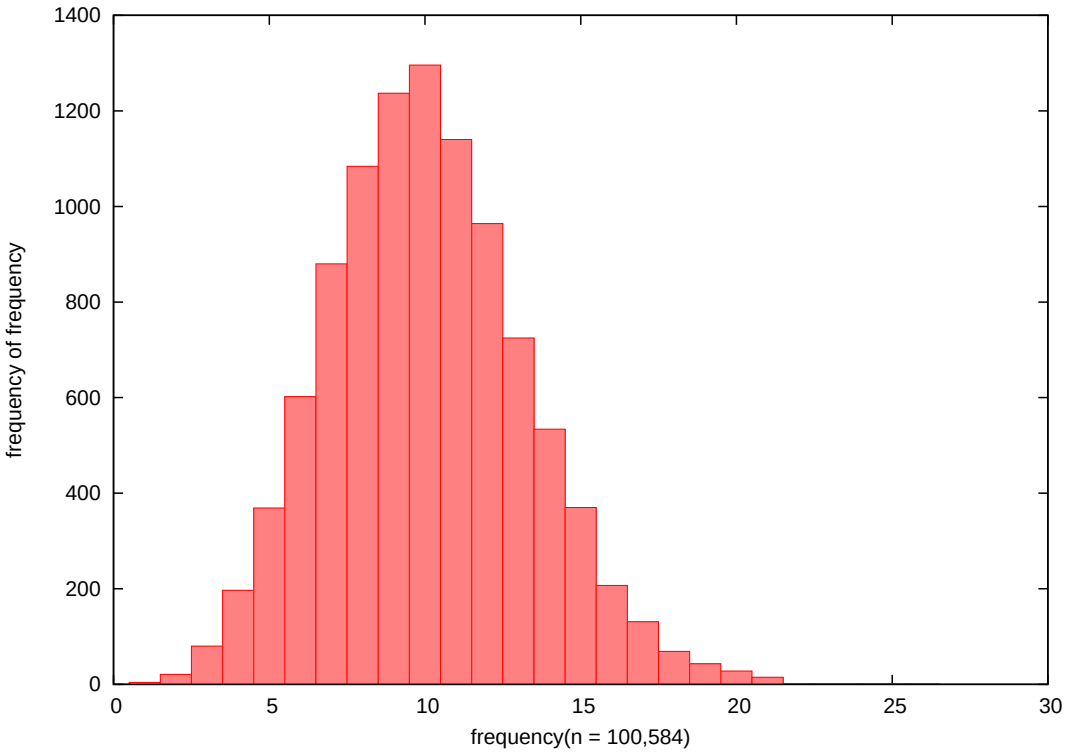

図 4.13: hash_sedgewick 関数の衝突耐性, 入力 URL 数：100,584[個]

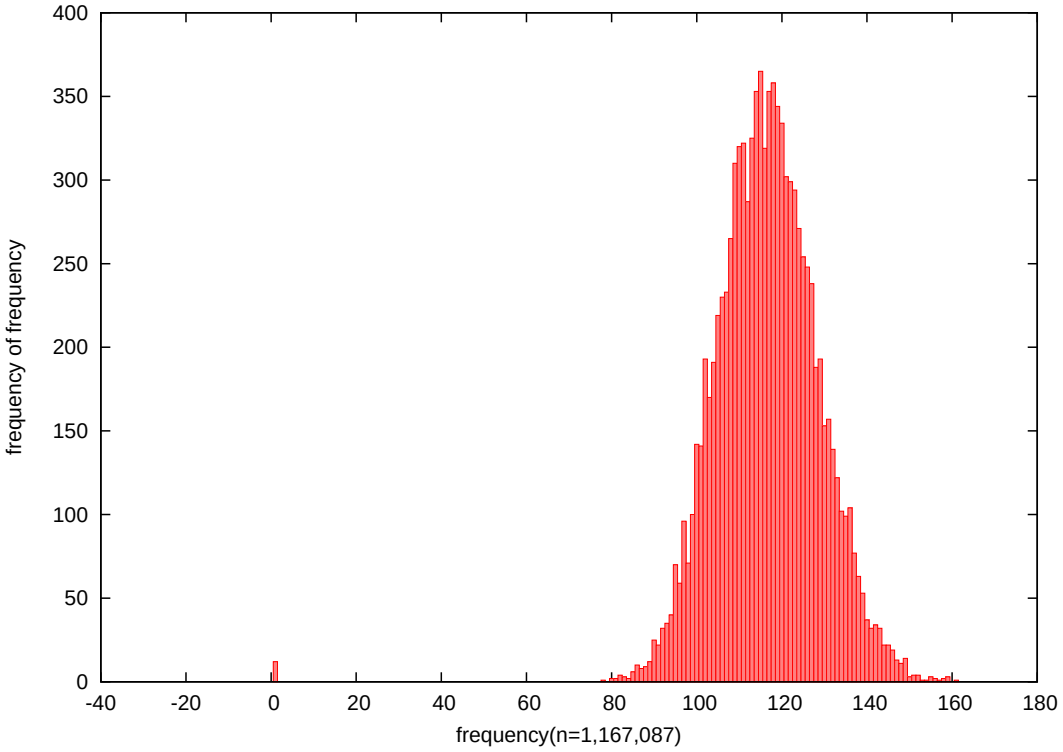

図 4.14: hash_sedgewick 関数の衝突耐性, 入力 URL 数：1,167,087[個]

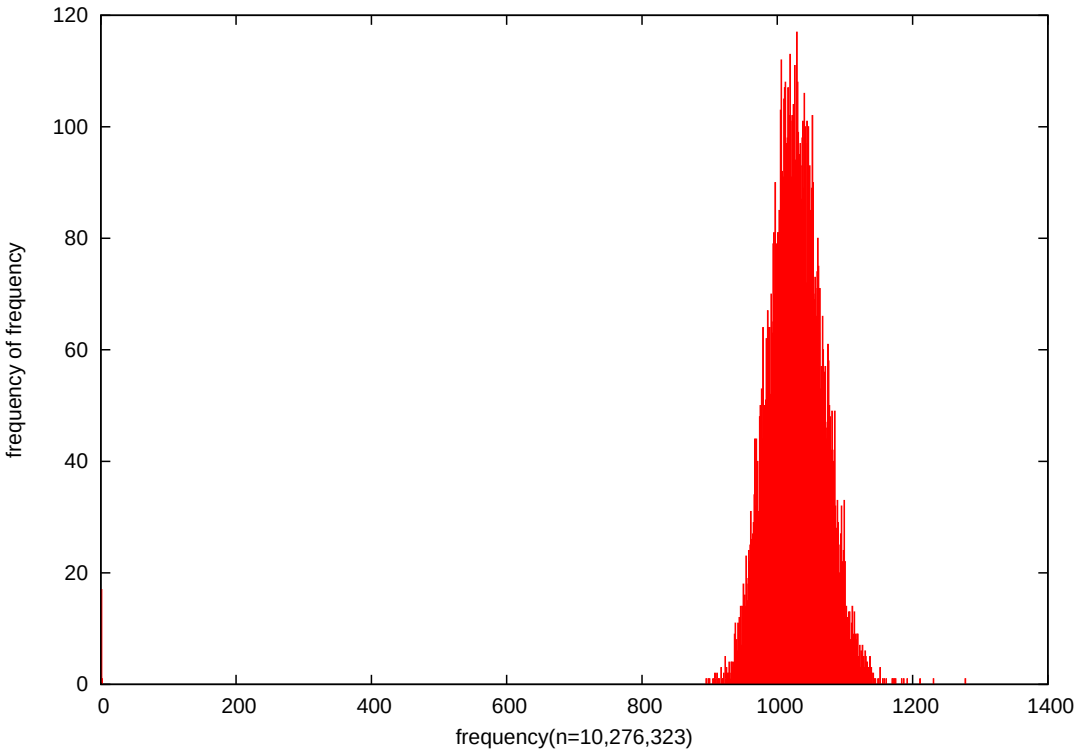

図 4.15: hash_sedgewick 関数の衝突耐性, 入力 URL 数：10,276,323[個]

4.2 学習データの収集と記録

4.2.1 学習期間の設定

検出システムが稀か否か判別するためには、事前にその基準となる平常時のアクセス傾向を知っておく必要がある。そのために、本研究では指定されたアクセス間隔に相当する期間以上を学習期間として観測データを事前に蓄積する。このことから、学習期間の長さは検出する URL の稀さ間隔を決定する重要なパラメータとなる。学習期間は検出するアクセス間隔以上を設定する必要があるが、それ以上の期間についてはサービス利用者の目的や運用予定のハードウェアの性能に応じて適切に設定すればよい。

4.2.2 学習データの保存と辞書データ

学習期間内に記録された観測データは、学習データとして学習終了後に二次記憶装置へ記録する。次に、検出期間へ移行すると学習データは新たに入力された観測データと混じり、まとめて一つの観測データとして取り扱われる。学習データの記録先のメディアは、Web-Prospector が動作しているハードウェアから物理的に分離されており、可能な限り信頼性の高いメディアが理想的である。学習データ及び観測データを二次記憶装置へ記録する際には、先に設計したデータ構造によってメインメモリ内に格納されたデータを図 4.16 のようなテキスト形式に変換したものを記録する。この形式は、先頭から「各 URL の最新のアクセス時刻」、「URL の文字列」、「アクセス間隔とその頻度を記録した n[個] のデータ」の順に並べられ、1[URL] あたり一行の可変長文字列となる。各値の間は、原則的にスペースで区切られるが、URL 文字列とアクセス間隔の頻度データとの間には URL 文字列との混同を避けるため、連続したスペースを 4 個と縦棒文字"|"が付加される。これが URL 文字列の終わりを示している。今後、本論文ではこのテキスト形式に変換されて二次記憶装置に記録されたデータを辞書データと呼ぶ。

検出期間中に不意の事故が発生してシステムを再始動させる場合に備え、定期的に観測データを辞書データ形式へ変換して二次記憶装置へ記録し、バックアップを取る。再始動させる際には、過去の観測データが保存されている辞書データからその区切り文字をもとにメインメモリ内へリロードする。図 4.17 に辞書データのサンプルを示す。

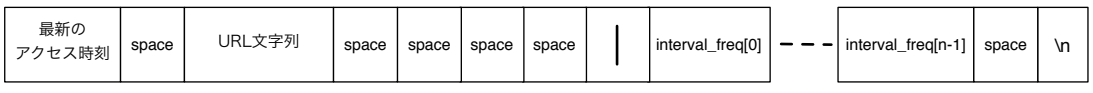

図 4.16: 辞書データの記録形式

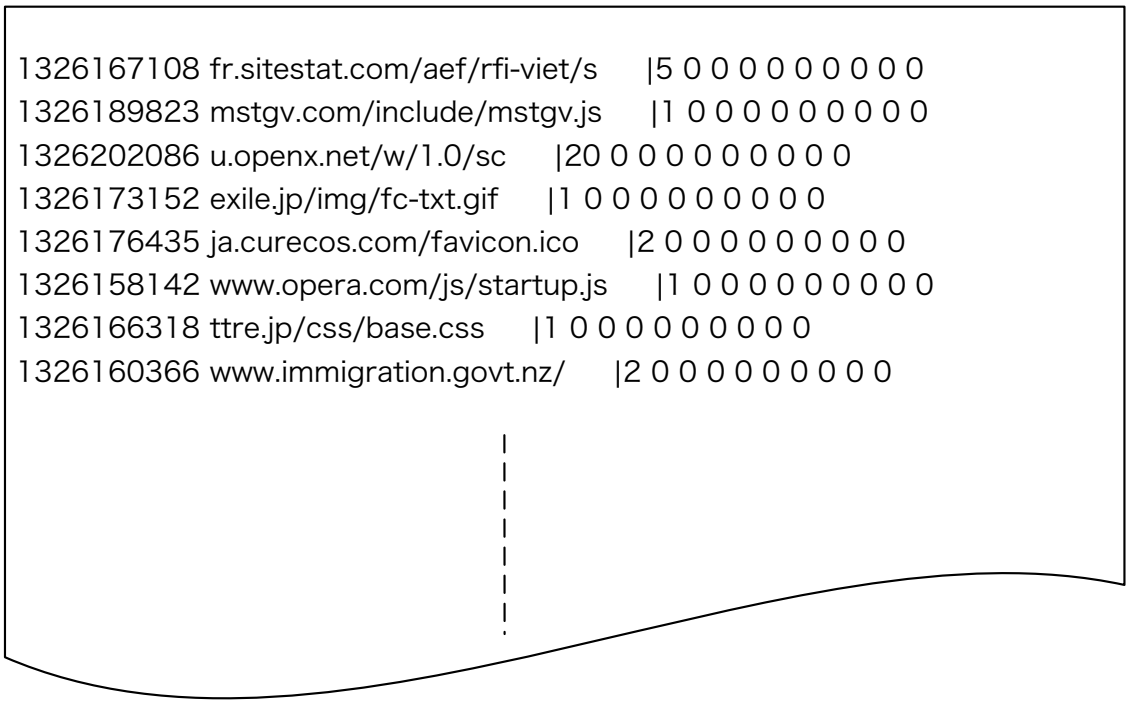

図 4.17: 辞書データのサンプル

第5章　URL Based Web-Prospectorによる検出結果とその評価及び考察

5.1　観測環境

4章で設計、実装したURL Based Web-Prospectorを用いて、実際に検出実験を行った。観測場所は、本学の学内ネットワークである。観測環境のネットワーク構成を図5.1に示す。URL Based Web-Prospectorは、学内ネットワークから隔離された計測専用のネットワークセグメントへ設置した。観測対象のパケットは、学内ネットワークからから学外へWebアクセスする際に送信されたWebサーバへのリクエストパケットである。入力パケットは、学内ネットワークから直接取り込むのではなく、図5.1のように学内と学外との境界に設置されたL2スイッチがミラーリングしたパケットを一度File ServerのNexusへダンプデータとして記録しておく。Web-Prospectorは、File ServerのNexusに記録されたパケットダンプを読み出すことでパケットをキャプチャする。また、定期的にバックアップのために書き出される辞書データや稀なURLが記録されたデータもNexusへ保存する。

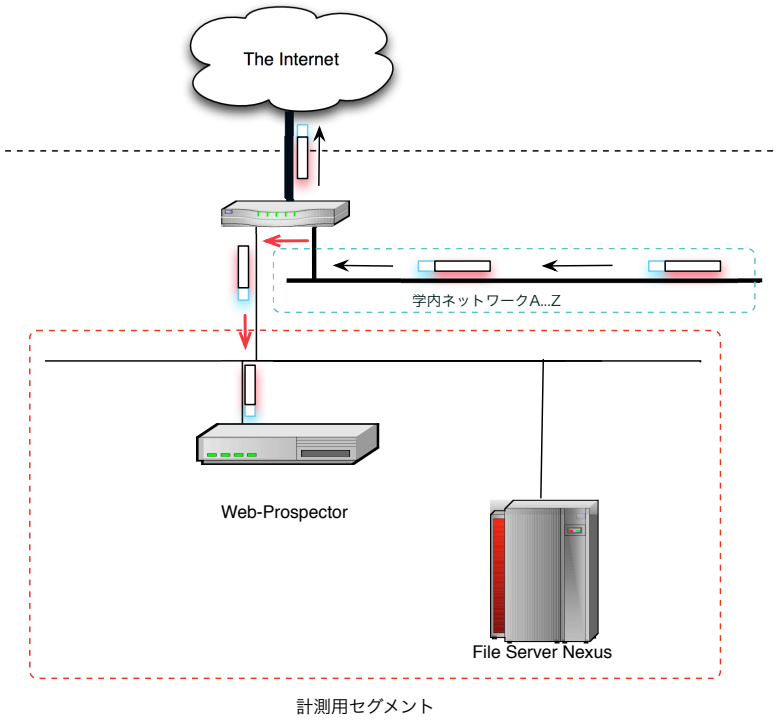

図5.1: 観測環境のネットワーク構成

5.2 観測実験の条件

観測実験は、16 日 (2012/1/10 ~ 2012/1/26) 分のパケットダンプを読み込ませて行った。本実験は、観測実験の中でも最初に行った実験である。そのため、本学学内ネットワーク環境において稀であると客観的に証明可能なアクセス間隔及びアクセス頻度が明確ではないため、まず式 5.1 と式 5.2 を判定条件として検出実験を開始した。各条件式における $THRESH\_TIME$ は、稀な Web ページへのアクセスであると判定するアクセス間隔の閾値を指し、$THRESH\_F\_max$ はアクセス頻度の閾値を指す。また、上記の 2 つのパラメータを軸として両条件式を満たす領域を図示したものが図 5.2 の青色で塗りつぶされた領域である。

$$THRESH\_TIME >= 12[hour] \tag{5.1}$$

$$THRESH\_F\_max <= 100 \tag{5.2}$$

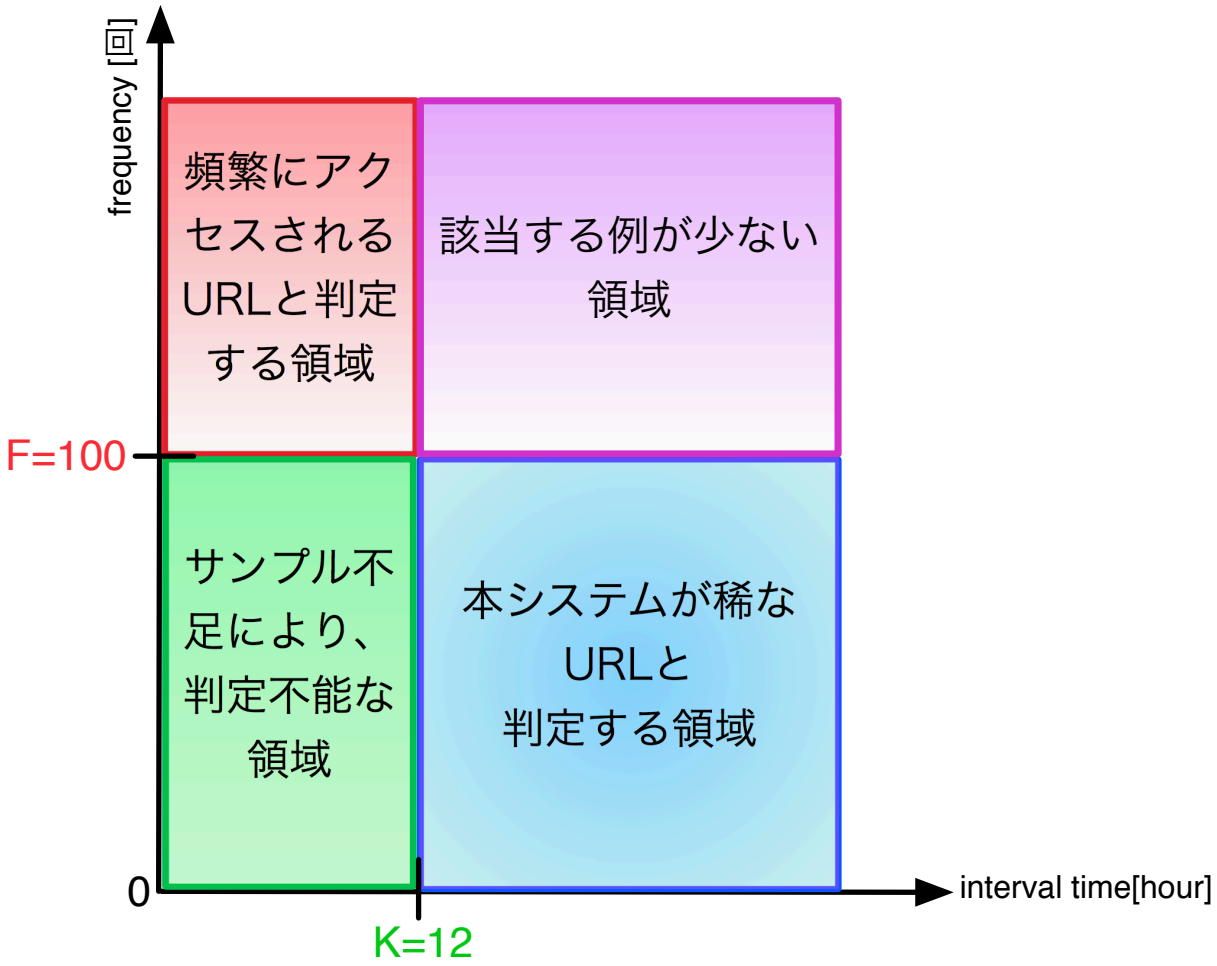

図 5.2: 本実験における稀な URL の判定領域

本学学内ネットワーク環境において 3 日間と 16 日間での Web サーバへのリクエスト数の分布を示した図 5.3 と図 5.4 から、Web アクセスには 1 日単位と 7 日単位の周期性があることが分かった。そのため、学習期間を 1 日とした場合と、7 日とした場合の 2 つのシナリオを用意して稀な URL へのアクセス数の累積値を計測する実験をした。それぞれの条件における観測点は、図 5.2 のように学習期間が 1 日の場合には、検出器間を 1, 2, 4, 6, 8, 14, 16 と増加させて観測した。学習期間が 7 日の場合には、検出器間を 8, 14, 16 日として各観測点を計測した。

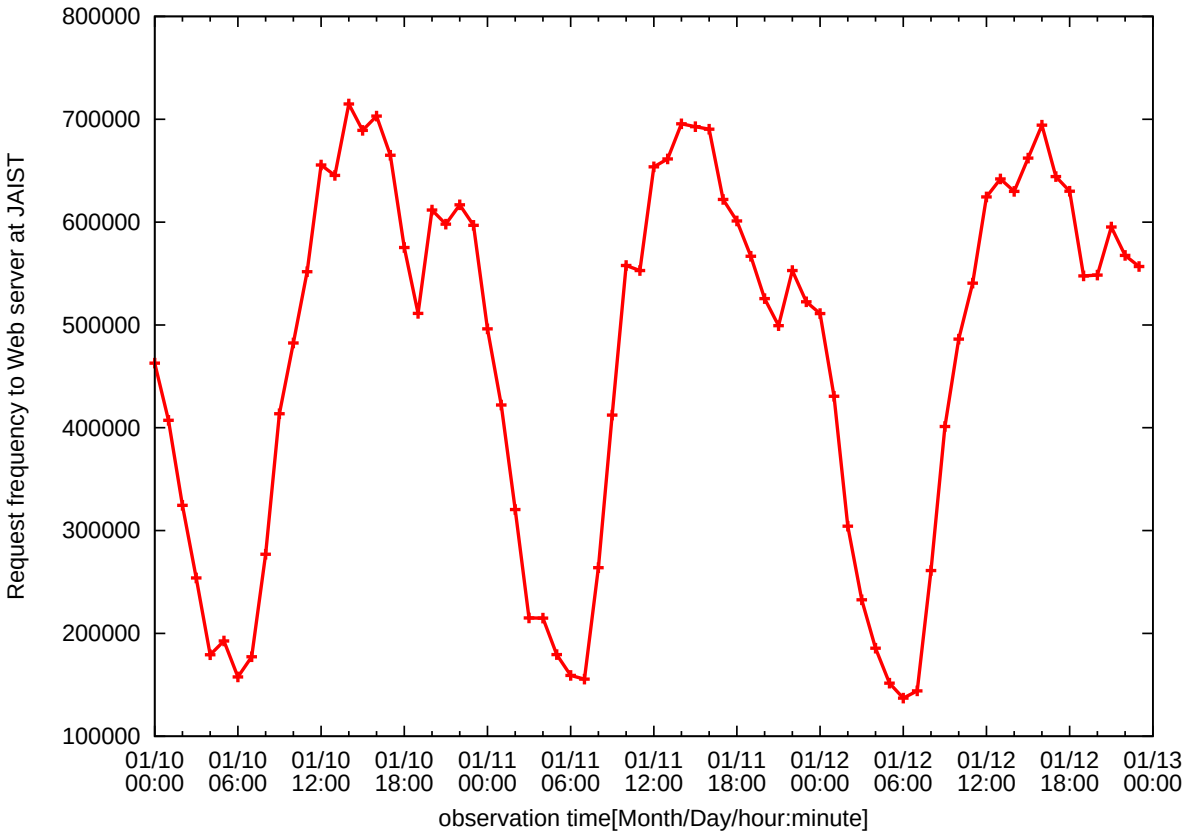

図 5.3: JAIST における Web サーバへのリクエスト回数の分布 (3 日間)

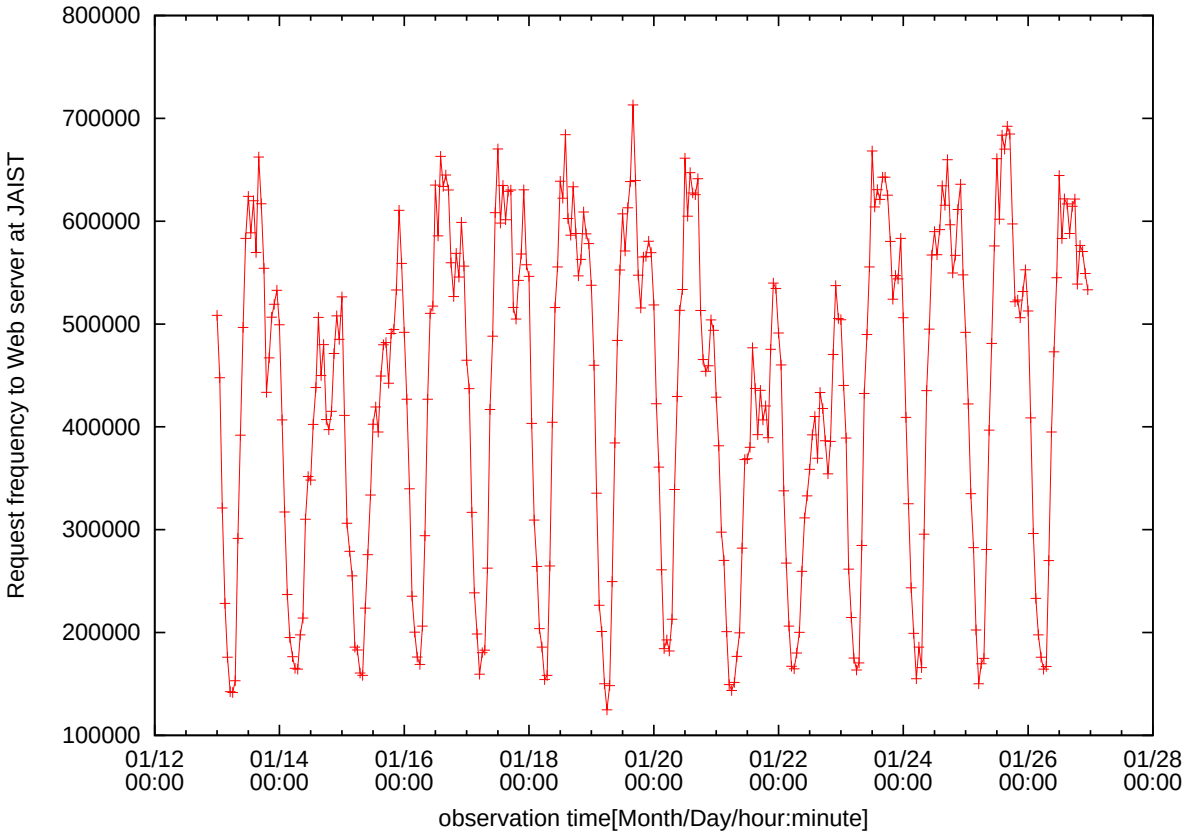

図 5.4: JAIST における Web サーバへのリクエスト回数の分布 (16 日間)

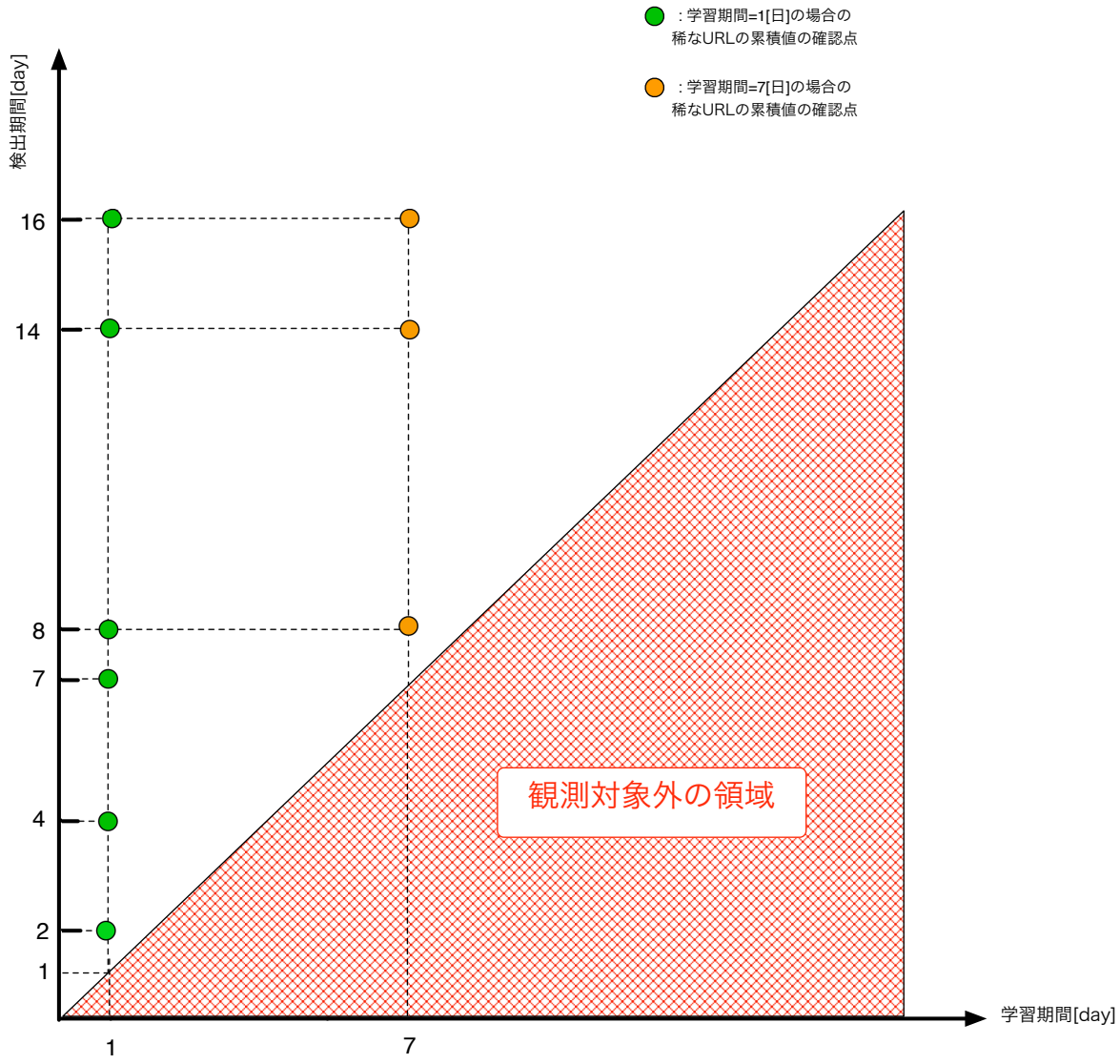

図 5.5: 学習データに応じた本実験における稀な URL の総数の確認点

5.3 システムパラメータの設定

4 章の設計内容をもとに、本観測実験のシナリオに最適なシステムパラメータを設定する。まず、本観測実験期間内に出現するリクエスト URL の総数を推定するために本学学内ネットワーク環境における任意の一日分の URL の総数を測定した。測定対象のデータは 2012 年 1 月 11 日 (水) に観測された本学学内ネットワーク上を流れているトラフィックデータである。その結果、2,519,744[個] を確認した。その個数から、ハードウェアに搭載されているすべてのメモリを使用する必要があると考えられるため、年間に出現が予想される URL の上限を 15,000,000[個] として設定した。下記に、観測システムのハードウェア構成を表 5.1、設定したパラーメータを 5.2 にまとめた。

表 5.1: 検出システムのハードウェア構成

機種名	Fujitsu PRIMERGY RX200-S6
CPU	Quad-Core Intel Xeon E5620 2.4[GHz] × 2[socket]
メイン・メモリ	48[GB](DDR3 1333 4[GB/slot] × 12[slot])
NIC	Intel 82575EB Gigabit Ethernet Controller(10BASE-T/100BASE-TX/1000BASE-T) × 2[port]

表 5.2: 本観測実験における Web-Prospector のシステムパラメータ

設定項目	値
ハッシュテーブルの長さ x:	150,000,000
配列 interval_freq の長さ n:	15
配列 interval_freq のデータ型	unsigned long(8[Byte])

5.4　学習データの収集と検証

本実験では、稀な URL の検出動作を始める前に学習期間を設定して学習データを蓄積し、それを学習終了後に辞書データに変換して二次記憶装置へ記録する。検出期間へ移行すると、始めに二次記憶装置に記録された辞書データをメインメモリ内へリロードする。読み込まれた学習データは検出期間移行後も新規に入力されるデータと区別なく引き継がれ、そのまま観測データとして取り扱われる。

検出実験の前に学習データが正確にメインメモリ内に記録されているか確認するため、URL をキャプチャしているフロントエンドプロセスが URL 文字列をバックエンドプロセスへ渡す際に、同時にその URL 文字列をログとして出力させた。そのログファイルを key として、バックエンドプロセスに記憶されたデータを検索し、該当するデータをすべて辞書データへ変換して保存した。このデータと既に辞書データ形式で記録されていた学習データを比較することで検証作業 (図 5.7) を行った。

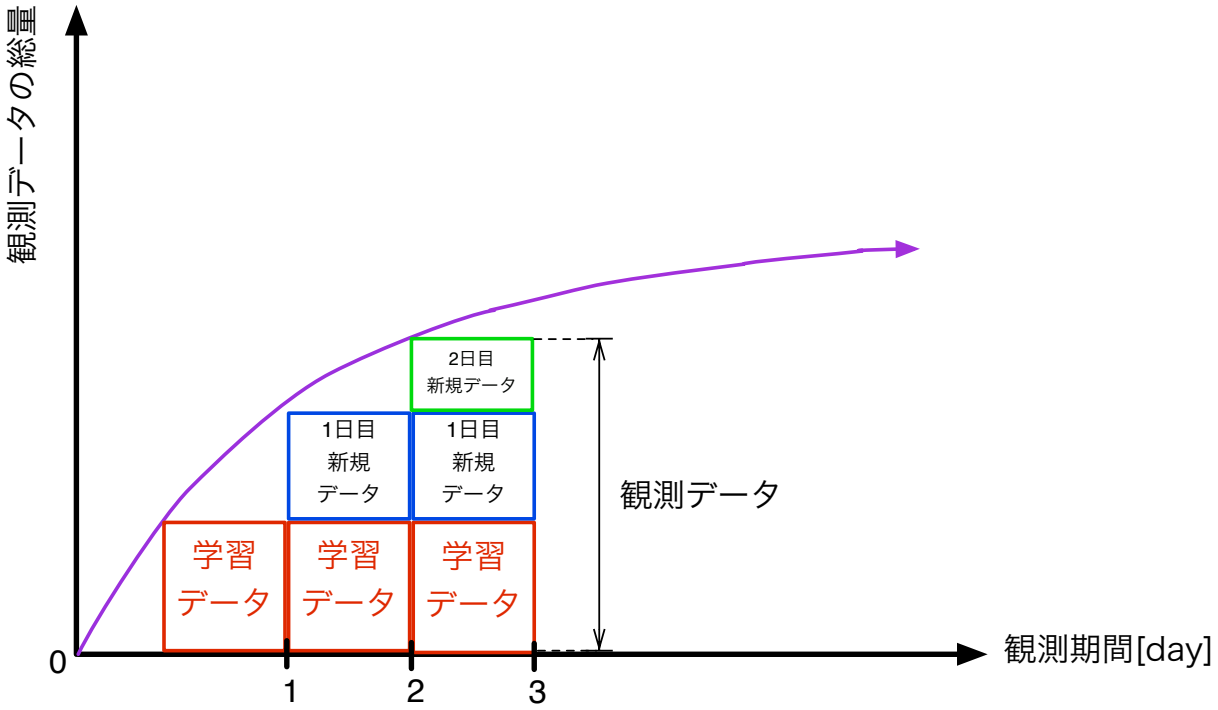

図 5.6: 学習データの引き継ぎ

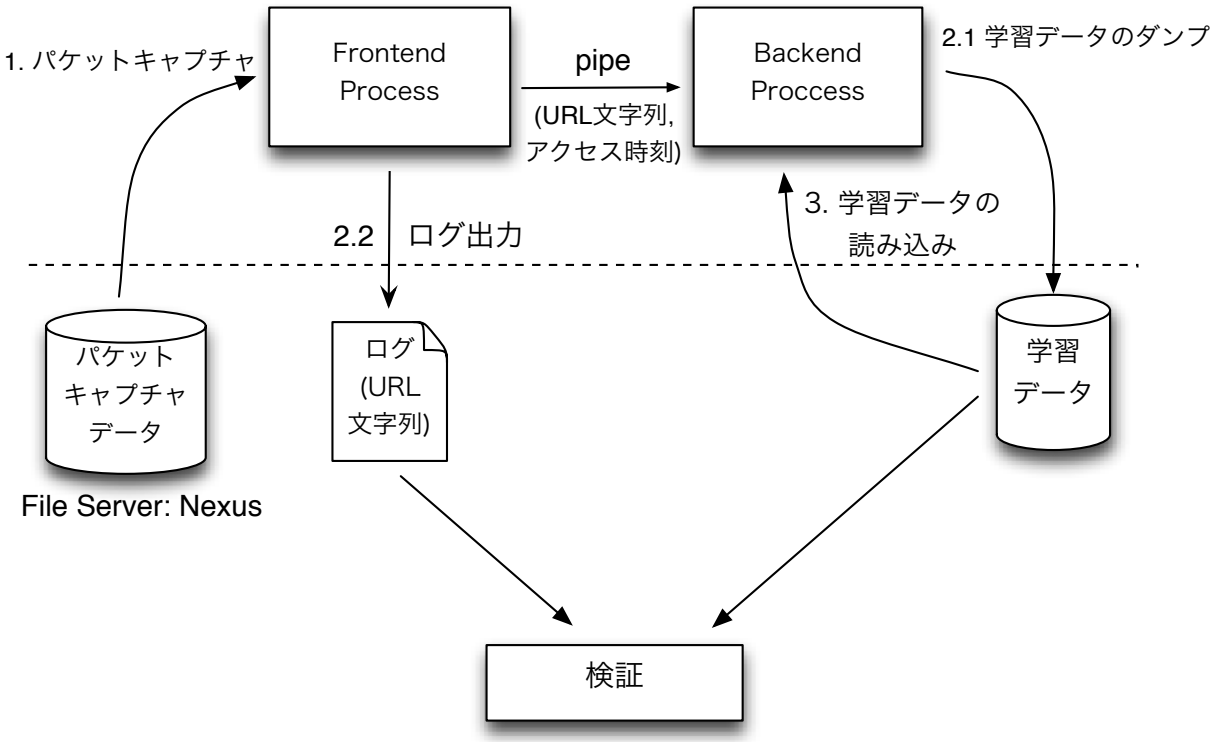

図 5.7: 学習データの収集と検証

5.5 検出結果と評価

5.5.1 観測されたリクエスト URL の総数

本実験によって得られた学内ネットワーク環境から Web サーバへリクエストされた URL の総数の分布を図 5.8、その両対数グラフを図 5.9 に示す。表 5.3 から学習期間 (0 日目) では 2,519,744[個] だったが、2 週間 (14 日目) の間にほぼ 10 倍になっていることが分かる。図 5.9 上の緑色の点線 f(x) は図 5.9 上の分布曲線を近似したものであり、ほぼ直線となることが分かる。この傾き γ を測定した結果、約 –1.77 であった。この値に最も近い結果が得られた研究例は、1997 年に 259,794 サイトに渡って分散された 5,000 万ページの Web ページを Alexa 社のクローラーを使って調査した文献 [4] の γ 値 (-1.94) である。値が多少前後している理由として、1997 年時点と 2012 年 1 月時点の Web アクセス環境では、閲覧するコンテンツの種類が異なること (動的な Web コンテンツなど) やインターネットへの接続環境が異なること (回線速度)、観測環境におけるユーザ数が異なることなどが一因と考えられる。

表 5.3: 16 日間の検出実験で観測された URL の総数

期間	観測時間 [day]	総 URL 数 [個]
学習期間	0	2,519,744
検出期間	1	4,859,787
	2	6,910,271
	4	10,154,205
	7	15,145,043
	8	16,896,819
	14	24,023,731
	16	28,765,877

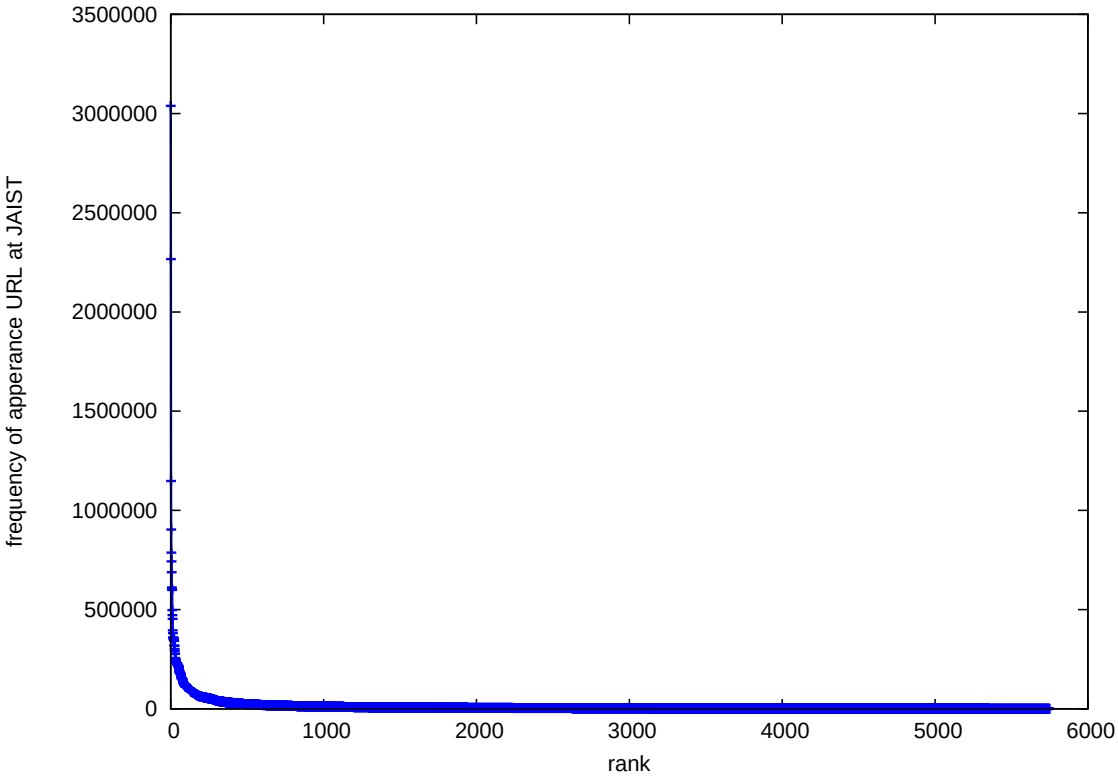

図 5.8: 本学学内ネットワーク環境において 16 日間に出現した Web サーバへのリクエスト URL の総数

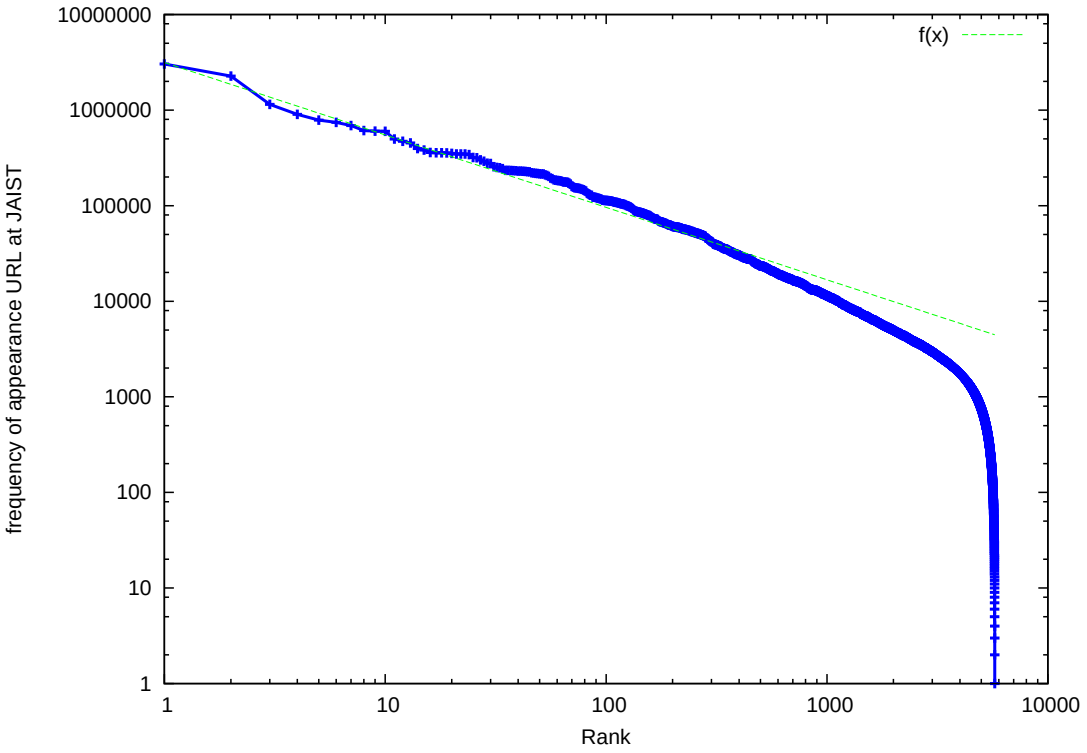

図 5.9: 本学学内ネットワーク環境において 16 日間に出現した Web サーバへのリクエスト URL の総数 (両対数軸)

5.5.2 検出された稀な URL の分布

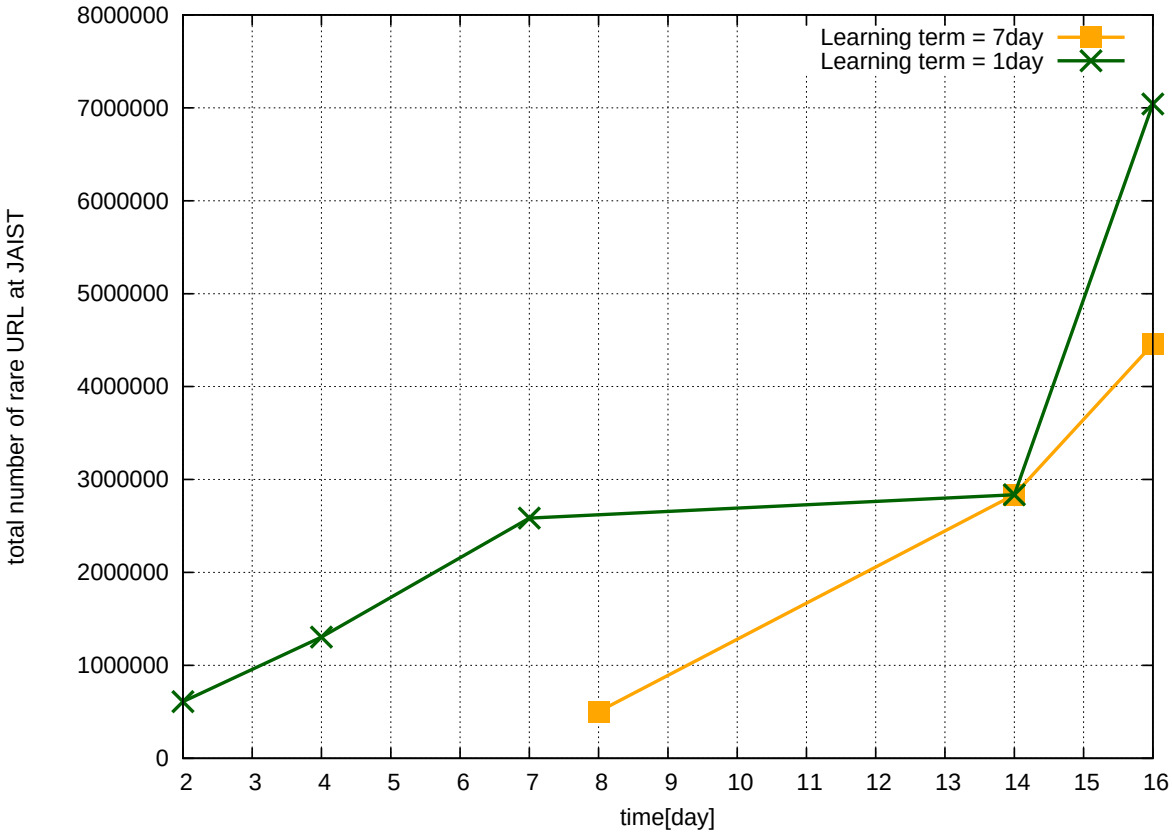

図 5.10: 稀な URL の検出結果

図 5.10 に、各観測点での稀な URL の検出数の累積値の分布を示す。このグラフをもとにして、学習期間の変化が稀な URL の検出総数に及ぼす影響を考察する。学習期間が 1 日の場合のグラフでは、7 日までは急激に増加しており、7 日以降は傾きが一時的に緩やかになっている。学習期間を 7 日とした場合では、検出期間に移行直後の 7 日間は学習期間が 1 日の場合の最初の 7 日間とほぼ同じ傾きが得られていることから、最初の 1 週間目と次の 2 週間目でリクエストされた URL はほぼ同じものである可能性が高い。そして、それが約 6 日間続いた後、6 日目以降は学習期間が 1 日の場合に比べてグラフの傾きが緩やかになっている。この理由として、7 日間の学習期間によって出現頻度が 100[回] 以下の URL が減ったこと、又は 100[回] に近づいた URL が増えたたことが影響していると考えられる。

5.6 URL Based Web-Prospector の辞書データサイズ及びメモリ使用量

16 日間の本実験において、提案システムは長期観測が可能であるか検証する。そのため、稼働中のシステムのメモリ使用量の最大値や二次記憶装置へ記録した辞書データのサイズについて学習期間が 1 日の場合の結果を図 5.4、学習期間が 7 日の場合の結果を図 5.5 に示す。なお、メモリ使用量の最大値は vmstat コマンドを実行することで取得し、辞書データのサイズは du コマンドが返したファイルサイズである。

表 5.4: 学習期間が 1 日の場合のメモリ使用量と辞書データサイズ及び処理時間

期間	観測時間 [day]	メインメモリの最大使用量 [MB]	辞書データのサイズ [Byte]	処理時間 (時間:分:秒)
学習期間	0	32,408	325M	00:19:44
検出期間	1	32,748	638M	00:40:40
	2	32,935	916M	01:39:52
	4	34,148	1.4G	02:34:46
	7	35,440	2.1G	04:22:16
	8	35,601	2.3G	02:10:11
	14	34,475	3.3G	11:18:11
	16	38,613	3.9G	07:59:44

表 5.5: 学習期間が 7 日の場合のメモリ使用量と辞書データサイズ及び処理時間

期間	観測時間 [day]	メインメモリの最大使用量 [MB]	辞書データのサイズ [Byte]	処理時間 (時間:分:秒)
学習期間	0	32,620	326M	00:19:31
	1	32,762	638M	00:40:14
	2	32,919	916M	00:57:16
	4	33,258	1.4G	02:07:21
	7	35,440	2.1M	08:31:18
検出期間	8	33,750	2.3G	02:07:40
	14	34,480	3.3G	11:15:16
	16	35,873	3.9G	08:01:52

メインメモリの最大使用量は 16 日目で 38G[Byte] に達しており、出現した URL の総数が 10 倍になるとメインメモリの使用量が約 1.2 倍になることがわかる。本実験期間内の増加率を保った

場合、提案システムのメモリ使用量は 1ヶ月後には 46,336M[Byte] となり、ほぼメモリ使用量が 100[%] になる。

URL 文字列以外のメンバによって消費されるメモリ使用量は、アクセス間隔の頻度を記録する配列:interval_freq が 8[*Byte*] × 25[個] = 200[*Byte*]、最新のアクセス時刻を記憶する変数が 8[Byte]、URL 文字列が格納されているアドレスを指し示すポインタが 8[Byte]、要素の使用中の有無を表すフラグが 1[Byte] となり、合わせて 217[Byte/URL] となる。これらは、プログラム中で静的に宣言しているため、本システム始動時に確保するハッシュテーブルの要素数 (本実験では 150,000,000[個]) 分だけあらかじめメインメモリ中に確保されることになる。よって、本実験では実験開始直後から式 5.3 より約 30.3G[Byte] 消費することになる。

$$217[Byte] \times 150,000,000[個] = 31042.1M[Byte] \approx 30.3G[Byte] \tag{5.3}$$

これに対して、URL 文字列を格納する為に消費した記憶量は式 5.4 から 4.7G[Byte] となり、その内訳は、1[URL] あたり malloc() によって必要となる管理領域が 8[Byte] と URL 文字列そのもののデータ量と考えられる。

$$35873M[Byte] - 31042.1M[Byte] = 4830.0M[Byte] \approx 4.7G[Byte] \tag{5.4}$$

この中で、実際に URL 文字列そのものが占める使用量は、式 5.5 より約 4.5G[Byte] となる。これを本実験で出現した全 URL の個数で割ると、1[URL] あたり平均 168[Byte] となる。

$$4830.0M[Byte] - (8[Byte] \times 28,765,877[個]) = 4610.5M[Byte] \approx 4.5G[Byte] \tag{5.5}$$

ここで、本システムがハードウェアに搭載されている 48G[Byte] のメインメモリの中から 46G[Byte] まで利用可能だと仮定すると、残り 11G[Byte] が使用可能となる。そのため URL 文字列と malloc() が用いる管理領域のために使用可能な容量は、式 5.6 から約 15.7G[Byte] となる。

ここで、本システムが 10 年間インメモリで稼働可能と仮定した場合、1ヶ月当たり URL(管理領域含む) の記録に使用可能な容量の平均値は、式 5.7 より約 134M[Byte](=836,364[個]) となる。

$$4.7G[Byte] + 11G[Byte] = 15.7G[Byte] \tag{5.6}$$

$$160768M[Byte] \div (12[month] \times 10[year]) = 133.9 \approx 134M[Byte/month] \tag{5.7}$$

これに対して、本実験期間内の増加率を 10 年間保ち続けると仮定して、10 年間で必要な記憶容量を試算する。まず、観測される URL の総数は、10 年後には約 121[億個] と推定される。アクセス間隔を 5 年間まで識別可能とすると、アクセス間隔 (interval_freq) の配列の要素数は、28 個必要となる。そこで、URL 文字列 (管理領域含む) 以外のデータによる使用量は、式 5.8 より約 26.5T[Byte] となる。URL 文字列の記録 (管理領域含む) によって消費される容量は、URL の平均長を本実験で

得られた 168[文字] とすると式 5.9 から約 19.4T[Byte] となる。そのため、これらを無圧縮でメインメモリ内に格納するには、全体で約 46T[Byte] 必要である。

$$241[Byte/URL] \times 121,000,000,000 = 26.52T[Byte] \approx 26.5T[Byte] \tag{5.8}$$

$$(168[Byte] + 8[Byte/URL]) \times 121,000,000,000 = 19.36T[Byte] \approx 19.4T[Byte] \tag{5.9}$$

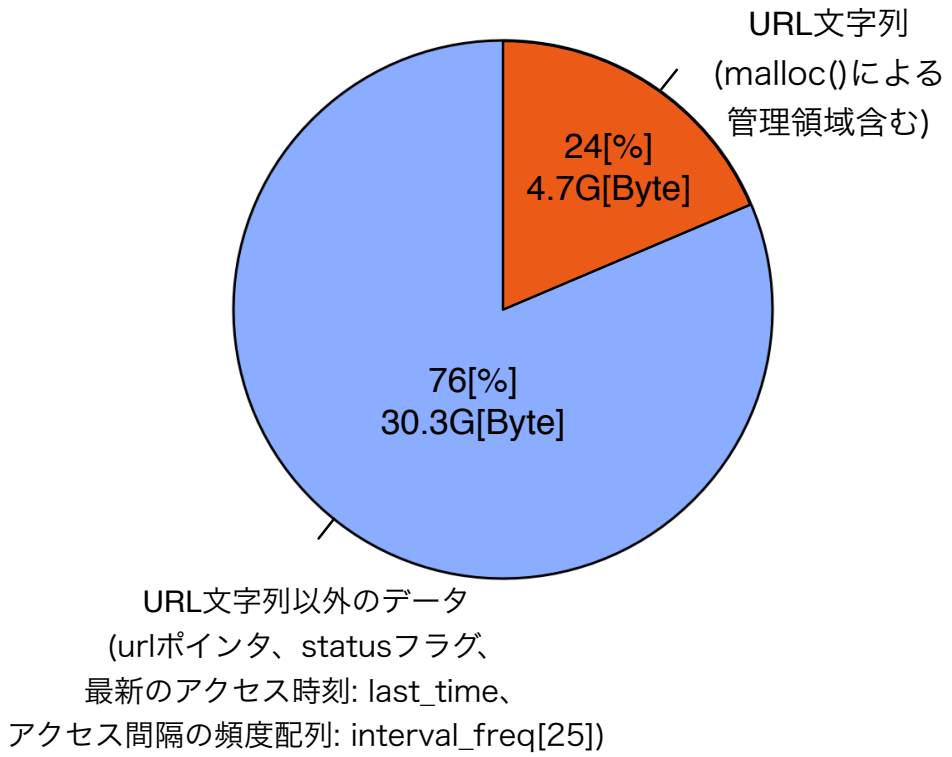

図 5.11: 本実験における URL 文字列 (管理領域含む) とそれ以外のデータ量の割合)

5.7 本システムの機能評価

本節では、本研究で掲げた検出システムに求められる要件に対して提案システムが実現した項目を検証し、不足分については次章「今後の展望」で述べる。3 章において、本研究の目的を達成するためにシステムに求められる要件は、大きく分けて 3 項目挙げられていた。

1. 長期観測 (最低 1 年以上) によって生じる膨大な観測データの取り扱いが可能であること
2. 実時間計測及び検出が可能であること
3. 計測期間に依存しない公正な検出手法を有していること

この 3 項目に対して機能評価を行う。

5.7.1 長期観測に対する耐性

本研究で目標としているシステムは稀にアクセスされる Web ページを検出するために、観測期間が長期となる検出システムである。さらに、実時間検出を実現するためにインメモリで処理する方式を採っている。長期観測のために本システムに求められ機能の一つは、長期間に渡ることによって生じる膨大な観測データの中から稀さを峻別可能なパラメータを抽出し、可能な限り記憶容量の低い形式に変換して記録する機能である。本研究では、稀さを峻別するパラメータとしてアクセス間隔に着目し、そのアクセス間隔を 2 の累乗ごとの範囲に区切ることでアクセス間隔を一定の区間ごとにランク分けした。同一範囲内のアクセス間隔はすべて同一ランクのアクセス間隔とみなして集約し、同一ランクのアクセス間隔の頻度を計測するデータ構造を設計した。例えば、観測期間が 10 年間 (= 311,040,000[秒]) である場合でも必要な配列:interval_freq の要素数は、29[個](2^{29} = 536,870,912[秒]) となる。アクセス間隔の最大値を 1 年と設定した本実験では、25[個] であったことから 1[URL] あたり 32[Byte] 増加するに留まる。アクセス間隔を一つ一つ細かく記録した場合に比べて、アクセス間隔を対数ベースでランク付けして集約することで記憶容量を大幅に削減することに成功した。また、本方式は同一ランクのアクセス間隔の頻度をカウントしていく方式のため、URL 文字列以外は新たに記憶領域を追加することはない。そのため、記憶容量を低く抑えられるだけでなく、設計時から稼働中のシステムのメモリ使用量の見通しを立てやすいため、システムの安定稼働に大きく貢献する方式である。

5.7.2 実時間計測に対する耐性

本システムは、ネットワーク上を流れるパケットを取り込みながら、16 日間の検出実験の中では観測した URL のアクセス間隔の算出やその頻度の計測を実時間で処理し、稀な URL の検出動作を行うことができた。しかし、本研究における稼働実験では、あらかじめ外部のファイルサーバに記録しておいた 16 日分のパケットダンプファイルを読みませただけであるため、長期間 (1 年〜) に渡って安定して実時間で処理可能であることを証明することができなかった。

5.7.3 計測期間に依存しない公正な検出手法

本システムは稀な URL へのアクセスか否か判定するために、最新のアクセス時刻と直近のアクセス時刻との差分であるアクセス間隔を判定用のパラメータとして用いた。そのため、前回のアクセス時刻を 1 つ記録しておけば、単体の Web アクセスに対して稀なアクセスであるか否か判別可能である。そして、アクセス間隔の頻度情報と組み合わせることで、その URL へのアクセスの稀さを別の観点からも評価可能となっているため、公正性の高い検出手法であるといえる。

表 5.6 に提案システムの達成度合いを項目別にまとめた。

表 5.6: 本方式のシステムの評価

評価項目	提案システム
稀なアクセスの検出アルゴリズム	○
実時間検出	○
長期運用 (メモリ使用量)	△
長期運用 (安定性)	△

第6章　今後の展望

6.1　動的に拡張可能なハッシュテーブルの必要性

本システムは、5 章の図 5.11 にあるように、URL 文字列以外のデータが全体の 76[%] を占めている。これによって、膨大なデータ量となる URL 文字列を格納するスペースを圧迫しており、現在の設計では 1 億 5 千万 [個] の URL を記録することは不可能である。そのため、URL 文字列以外の未使用リソースを減らす為にシステム起動時にそれらを静的に宣言するのではなく、観測された URL の数に応じてハッシュテーブルを動的に拡張する必要がある。これによって、記録可能な URL 数を増やすことが可能となり、メインメモリを最大限に活かすことができる。

6.2　URL 文字列の圧縮

本システムは、URL 文字列を key として、ハッシュ法によって該当するバケットを検索するシステムである。URL は、Web ページのコンテンツ一つ一つに割り当てられているため、親ディレクトリやホスト名が共通である可能性が高い。そのため、パトリシアトライによる文字列の圧縮効果が期待できる。しかし、すべての URL を一つのトライ木で扱う場合、扱う URL 文字列の種類が多様になるために枝が深くなり、ツリーを構築するためのリソースや探索効率が大きく落ちることが予想される。そのため、すべての URL を一つのトライ木に格納するのではなく、コンテンツの種類が多い同一ホスト名の URL 群がツリーの構築対象として適している。そのため、二段階のハッシュテーブルを用いた記録、探索法を提案する。この手法は、URL をキャプチャした後にそのホスト名を key としてハッシュ関数に掛け、一段目のハッシュテーブル上の該当するバケットへアクセスする。もしそのホスト名で始まるパトリシアトライが構築されていれば該当するツリーの先頭へ跳ぶ。もし存在していなければ URL 文字列に含まれるパス名を key として再ハッシュし、2 段目のハッシュテーブルの該当するバケットへアクセスして、そこへアクセス間隔や最新のアクセス時刻などのデータを格納する方式である。

6.3 稀なアクセスの定義の拡張

本研究では、アクセス間隔がサービス利用者にとって稀であると定めた期間より長く、かつ一定のアクセス頻度よりも低い URL へのアクセスを「稀な URL へのアクセスである」と定義した。稀な URL へのアクセスモデルは本モデルだけではなく、長期間一定周期でアクセスされる URL に対して一時的にその周期が大きく乱れるようなアクセスが発生した場合、その URL に対して稀なアクセスが発生したともいえる。例えば、突発的にアクセスが集中したり、逆に完全にアクセスが途絶えるような事象が発生した場合である。現在のシステムでは、このようなアクセスを検出することができない。このタイプの稀なアクセスを検出するには、アクセス頻度やアクセス間隔だけでなく、長期間に渡ってアクセス周期やアクセス間隔の分散などを計測し続け、それらをを評価対象として判定する新たな機構を設計する必要がある。

6.4 二次記憶装置との併用

本研究で対象とする稀な URL へのアクセスは、めったに発生することがないため、稼働中の計算リソースの大半は出現頻度が高い URL を処理することに向けられていると考えられる。そのため、新規に出現してから長期間アクセスされていない Web ページのアクセスデータをを積極的に二次記憶装置へ追い出し、アクセスされる確率が高い URL に関するデータを優先的にメインメモリへ残す機構が有効である。また、メインメモリと二次記憶装置とのスループットの差を埋める為に、パケットのペイロードから検出操作に必要なデータだけ取り出し、取り出したデータを一時的に貯めておく専用のバッファを用意したり、出現した URL をハッシュテーブルの中から探索し、記録データを更新する処理をマルチスレッド化することでメインメモリ中に存在する URL を優先して同時並行的に処理し、二次記憶装置へ退避している URL に関する情報へアクセスする際に発生する待ち時間を有効に活かす工夫が必要である。

第7章　おわりに

本研究では、稀な Web ページへのアクセスを検出するために、膨大な数に及ぶインターネット上の Web ページを URL として識別し、稀にアクセスされる Web ページを検出するために、長期間に渡って膨大な観測データを取り扱う機構を実現することが課題であった。それを解決するための記録手法としてアクセス間隔に着目した。そして、アクセス間隔とその頻度情報を効率的に集計する機構を設計して、長期観測によって生じる膨大な記録データを有限の計算機リソースの中で実現する一手法を示した。そして、設計したデータ構造をもとに検出システムである Web-Prospector を実装し、実際に本学学内ネットワーク上でシステムの動作検証を行った。検出実験では、実際にネットワーク上を流れているパケットを入力する形ではなく、事前に File Server へ記録した 16 日分のパケットのダンプデータを入力ファイルとして読み込ませて検証した。その結果、短時間 (16 日間) であれば実時間で稀な URL を検出するシステムを実現できたことを確認した。

今後の課題は、実際に観測システムを長期間に渡って稼働させることでシステムの安定性を検証することや、動的に拡張可能なハッシュテーブルの実現、パトリシアトライを用いて URL 文字列を圧縮することが挙げられる。また、様々な条件のもとで検出実験を行い、観測環境ごとに稀な URL と判定できる出現頻度やアクセス間隔のしきい値を算出して稀な Web ページへのアクセスパターンを分類することが重要になる。それらのパターンに潜む共通点が明らかになれば、それをもとに様々な観測環境で同一の判定条件の元で本システムを稼働し、相互に比較することが可能になると考えている。その結果から、Web アクセスに潜む新たな知見を得ることができると期待される。インターネットが社会インフラとなった今日では、このような少数のアクセスも見逃さずに詳細に解析することは安定したネットワークを構築する上で無視できない要素であるといえる。

謝辞

本研究を進めるにあたり、様々な場面で多くの方々から多大な御助言や御助力をいただきました。それらの方々のご協力がなければ、本研究は成り立ちませんでした。ここに、心から厚くお礼を申し上げます。

研究を行うにあたり、主指導教官である知念賢一特任准教授には多大な時間を割いて多くのご助言とご指導をいただきました。ここに、心から深く感謝いたします。

本研究を始めるきっかけを与えてくださり、研究内容についても多大なご指導を賜りました主テーマ審査委員の篠田陽一教授に深く感謝いたします。また、本研究に関して適切な御助言をいただきました主テーマ審査委員である敷田幹文准教授、副テーマ指導教員である二木厚吉教授に深く感謝いたします。

独立行政法人情報通信研究機構 北陸 StarBED 技術センターの三輪信介センター長には、構築したシステムの実装方法や得られた実験結果について適切な御助言と御指導を頂きました。深く感謝いたします。情報社会基盤研究センターの宇多仁助教と小原泰弘助教 (現 カリフォルニア州立大学サンタクルーズ校 ストレージシステム研究センター) には観測実験を行うための環境構築について多大な御助力を頂きました。心より深く感謝いたします。

篠田研究室の OB である高野祐輝先生、博士後期課程の井上朋哉氏、安田真悟氏、明石邦夫氏、Muhammad Imran Tariq 氏、Latt Khin Thida 氏、Nguyen Lan Tien 氏には、研究の方針に関して活発に議論する機会を頂き、適切なご指摘とご指導を頂きました。改めて深く感謝いたします。

篠田研究室の博士前期課程の川瀬拓哉氏、鍛冶祐希氏、山田悠介氏、田部英樹氏、村上正太郎氏、大野夏希氏、向井雄一郎氏には研究生活を送るにあたり、様々なご意見、ご協力を頂きました。ここに深く感謝いたします。

最後に、研究生活を支えてくれた家族へ心から感謝いたします。

参考文献

[1] programming with pcap. http://www.tcpdump.org/pcap.html.

[2] Mawi working group. http://www.wide.ad.jp/project/wg/mawi.html.

[3] Squid. http://www.squid-cache.org.

[4] Lada A. Adamic and Bernardo A. Huberman. Power-law distribution of the world wide web. *Science*, Vol. 287, pp. 2115 – 2115, 2000.

[5] R. Albert and A.-L. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, Vol. 74, pp. 47 – 97, 2002.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, RFC2616. June 1999.

[7] M. Mealling, Ed., R. Denenberg, and Ed. Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations, RFC3305. August 2002.

[8] 増田直紀, 今野紀雄. 複雑ネットワーク. 近代科学社, 2010.

[9] Tim Berners-Lee(高橋徹 訳). Web の創成. 毎日コミュニケーションズ, 2001.

[10] Robert Sedgewick(野下 浩平/星 守/佐藤 創/田口東=共訳. アルゴリズム C・新版. 近代科学社, 2004.