ABSTRACT

CHEN, KUNSHENG. An Open Solution To Discover The Graph Structure Of World Wide Web. (Under the direction of Dr.Vincent W. Freeh).

The World Wide Web is a large complex network of inter-linked web pages. Understanding this structure is of immense benefit both economically and socially. Currently, there is incomplete or sparse information about the graph structure of the Web in the public domain. The full data is closely-guarded by a handful of corporations.

Nevertheless, studies on the topological structure of World Wide Web benefit not only scientists and e-commerce merchants but also common users. A better understanding of such a structure helps scientists to develop new technologies to improve the Internet. It also assists companies to build optimal e-commerce solutions to fulfill their business needs.

The goal of this thesis is to evaluate an open source solution to mapping the structure of the Web. In support of this thesis, we have implemented a prototype using existing open source software including voluntary computing library BOINC (Berkeley Open Infrastructure Network Computing) and Hadoop MapReduce framework. We utilize the computing power and disk space from BOINC to perform data collection and Hadoop MapReduce framework to perform data analysis on a large set of data..

Contribution of our research includes a low-cost open solution of a distributed web crawling system using BOINC and a URL ranking system utilizing Hadoop MapReduce framework. We also provide a feasibility study on crawling the web using the above solution and present experimental results.

An Open Solution To Discover The Graph Structure Of World Wide Web

by
Kunsheng Chen

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2010

APPROVED BY:

Dr. Frank Mueller	Dr. Xuxian Jiang

Dr. Vincent W. Freeh
Chair of Advisory Committee

DEDICATION

For my family who offered me unconditional love and support my Master study in North Carolina State University.

ACKNOWLEDGMENTS

I want to thank my advisor Vincent W. Freeh for his efforts and patience on guiding my research so far. Also thank Sergey Lyakh who initialized project Anansi. In addition, a thank you to Tyler Bletch for his great ideas and system support.

I would like to thank my committee members, Dr. Frank Mueller and Dr. Xuxian Jiang, who spent time and efforts on reviewing my thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Studies on the topological structure of World Wide Web benefit not only scientists and e-commerce merchants but also common users. A better understanding of such a topological structure helps scientists to develop new technologies to improve the Internet. It also assists companies to build optimal e-commerce solutions to fulfill their business needs, especially on advertising and search engine optimization. Common web users also benefit from the knowledge in the term of data acquisition and comparison.

Although the importance of the Internet's topological structure has attracted concerns from the public for a while, its detailed knowledge are still inaccessible to common users due to many reasons. Some e-commerce merchants such as search engines companies usually owned more concrete information of such a topological structure compared to the general public. Popular search engines such as Google and Alexa utilize web crawlers to keep collecting and updating their knowledge of the web graph, and usually provides lists of information by some kind of orders that are claimed to be correct and unbiased [15]. However, due to the business purposes, detailed information of such a graph structure have never been revealed, let along its correctness and integrity. Moreover, the way used to present data to web users are usually biased due to business factors even they are claimed not to be. In order to find out more details as well as provide open-source knowledge on such a topological structure, we design our own web crawling system to collect and validate related data.

Building a web crawling system is usually expensive and time-consuming as data needs to be collected, stored, updated and coordinated frequently. To collect information

through the web, a system has to uses programs called web crawlers to traverse the web cooperatively [15] and repeatedly, which is time-consuming due to the network traffic as well as data parsing, analysis, a buffering. Because of the frequent changes occurring in World Web Wide, the system also need to re-crawl and update existing pages periodically, which requires huge storage for encountered data as its size exposes. To better study the internal relationships between different or similar kinds of data, the system usually performs analysis on stored information by executing sets of programs. Those tasks definitely require sufficient memory and computing power. The above requirements are also part of reasons that keep common users from understanding the web graph. One possible low-cost solution to implement such a crawling system is utilizing voluntary computing, which is also known as peer-to-peer computing or global computing. Voluntary computing provides a feasible solution to gather computing resources from internet-connected computers over the general public [1]. The cost to utilize resources in voluntary computing is trivial as all participants are volunteers and willing to contribute their additional resources to scientific projects. The computing power of those resources is also huge and keeps rising as web users are increasing, which provides a large processing rate that is comparable to conventional supercomputers.

1.1 Proposed solution

The first step to study the topological structure of World Wide Web is to simplify it into a directed graph. The simplification of this topological structure provides us a direction to understand its knowledge as well as convenience to verify existing strategies used by e-commerce merchants.

1.1.1 The web graph

The web graph is a topological representation of World Wide Web, in which vertices and edges are identified as web pages and hyperlinks respectively [16]. As we know, World Wide Web is a collection of millions of web pages, each of which can be identified by a Uniform Resource Identifier (URI). In this case, World Wide Web can also be simplified as a directed graph, assuming web pages as nodes and each pair of connected pages have a directed link between them. As one of the most popular software systems to perform data acquisition on the web graph, searching engines are playing an important role in traversing

web content through World Wide Web, especially on the field of data collection and analysis [17]. In general, search engines provides a list of URLs related to certain information sorted by priority, which is conducted by a set of algorithms. One of the most famous ranking algorithm known to be used by well-known searching engine Google is PageRank [5], which was invented by one of Google's founders Larry Page.

A better understanding of the web graph provides the very first step to understand the whole web, based on which further research can be done to analyze the correctness and feasibility of the technology used in commerce. Validating the integrity claimed by those merchants also becomes possible. Moreoever, common users or small business can benefit from that knowledge when developing their own e-commerce solutions on the aspect of adapting to the web and attracting more business.

However, the study of web graph is really limited to how it is conducted and the value of data it provides. Although many research projects have been done to find out properties on such a graph and tried to assure integrity, biasing on data collection and analysis is unavoidable. Dynamic changes occur in web World Wide Web also make it difficult to maintain the correctness and availability of information in such a graph.

1.1.2 Voluntary computing

The computing power and disk space belonging to a network of personal computers can be gathered to solve problems that are too complex for a single-centralized system. Voluntary computing, also known as public-resource computing or global computing, was first introduced to the public in 1990s with two projects, GIMPS and Distributed.net. The goal of Voluntary computing is utilize those existing computing power distributed in Internet-connected personal computers around the world. Some of the successful projects emerged in late 1990s such as SETI@home, Rosetta@home and climateprediction.net have attracted more millions of participants, providing a couple times of processing rate as those largest conventional supercomputers in the world [1].

BOINC (Berkeley Open Infrastructure for Network Computing) infrastructure provides a software solution for researchers to create and operate voluntary computing projects. BOINC is running on a network of workstations owned by public volunteers and support distributed computing by utilizing CPU resources from each involving client [7]. It provides not only an open-source framework for software engineers to develop distributed system,

but also a user-friendly platform for volunteers to contribute their additional resource to those projects.

The potential computing power in the general public is huge and keep increasing, with the rapid steady growth of the number of personal computers [2]. Most of the voluntary computing projects are open to the public and is distributed over a large collection of machines, which make them easy to scale up by adding low-cost workstations or voluntary participants.

1.2 Implementating the system

In order to gather enough information over the internet efficiently as well as update and analyze collected data, an appropriate solution for the system should consist of at least two parts: the crawling system and the scheduling system. The crawling system is responsible for traversing web pages and collecting specific information that is required to draw the web graph. On the other hand, the scheduling system should have two subsystems: the URL ranking system and the scheduler. The URL ranking system analysis data collected by the crawling system and responses with a schedule on crawling and re-crawling, while the scheduler coordinates between the URL ranking system and the crawling system by handling data transition, compression and validation.

As mentioned above, the solution utilizing Voluntary computing framework BOINC [7] provides researchers an easy way to gather distributed computing resources over World Wide Web. The number of participants in voluntary computing is huge and increasing, indicating a growing computing power and available storage behind those related projects. The distributed web crawling system designed in our research can use resources from those volunteers by implementing the sub crawling system running on millions of voluntary clients and maintain a BOINC server that is acting as the scheduling system.

1.3 Contribution of our research

We are utilizing existing open source libraries such as BOINC, Hadoop to build a distributed web crawling system. The contribution of our research consists of following parts:

1. **Distributed crawling system running on BOINC**. As mentioned before, a web crawler is a program that explores World Wide Web by traversing web contents automatically and continuously. In our design, the crawling system is distributed and running on a network of BOINC clients from general public. Each voluntary client crawls the web independently and directly reports to the scheduling system built on BOINC [7].

2. **The scheduler built on BOINC**. The scheduler acts as a role for communication between client machines and the URL ranking system, which is responsible for dispatching new tasks and assimilating finished tasks from clients as well as controlling URL ranking system to perform data analysis. The scheduler is built on a BOINC server and is running a collection of daemons to perform the tasks [7].

3. **The URL ranking system built on Hadoop DFS**. The URL ranking system plays an important role on providing priority information for re-crawling tasks. We build the ranking system on Hadoop DFS, utilizing MapReduce programs to handle a large set of data collected by the crawling system [21]. The URL ranking system takes input from the scheduler and output information on priority for collected URLs.

4. **Feasibility study**. The crawling system is implemented using BOINC and Hadoop DFS, both of which can be scaled up easily by adding low-cost workstations. Detailed studies on the feasibility of the crawling system are discussed in chapter 5, in which estimations on system scaling are covered.

1.4 Organization of the paper

In this paper, we describe the software architecture of our distributed web crawler, which is built on existing open source software systems including BOINC and Hadoop. The project is named textitAnansi, which originally came from a character in West African and Caribbean folklore: Anansi is a spider but often appears and acts as a man, which is also known as Anase.

We present the design and model of our system, discuss the accuracy, performance and encountered problems. We also report the experimental results based on the collected data.

The paper is organized as below: chapter 2 reviews related works including previous studies on web graph, distributed web crawlers, BOINC infrastructure and Hadoop DFS. Chapter 3 introduces software architecture of the system. Chapter 4 presents detailed implementation of the model. Chapter 5 reports experimental results and evaluation of accuracy, performance and other encountered issues. Chapter 6 concludes and discusses future work.

Chapter 2

Related Work

In this section, we present previous work related to our study of the graph structure of World Wide Web such as how to map the web into a graph and some existing distributed web crawlers. We also introduce the existing technologies we used in our solution, which contains voluntary computing and MapReduce framework.

2.1 Mapping the web

Early studies on the topological structure of World Wide Web focused on a huge directed graph whose nodes are mapping to web pages in Internet with edges indicating connections between different pages. Those studies usually concentrated on measurement of degree values of web pages, in which in-degree and out-degree are defined as number of incoming and outgoing links connecting to the page, respectively [16].

Duncan Watts' team defines a rigorous way to estimate chain lengths between individual nodes, whose findings provide mixed support for the algorithmic hypothesis: all ordinary individuals can find short paths in a large social network besides pairs that are connected. Their experimental results also provide evidence that supports the 'six degrees of separation' assertion [10].

Several research teams have developed efficient link-based ranking algorithms based on the above web graph such as SALSA and HITS. Microsoft research team implemented the Scalable Hyperlink Store (SHS), which is a distributed in-memory database used to store information of the web graph and also serve as a general-purpose system for performing

ranking algorithms on the web graph.

2.2 Distributed web crawlers

The popular search engine, Google, uses distributed web crawlers to perform data acquisition [15]. The distributed system was implemented in Python and consisted of a URLserver and a number of high-performance web crawlers, each of which can crawl roughly 100 web pages per second.

Cho's team implemented a parallel web crawler system. They defined a distributed crawler as a web crawler with simultaneous crawling agents [6]. Based on this definition, their UbiCrawler was presented as a fully distributed crawler with crawling agents, each of which may run on a different computer. A limited number of threads are run in parallel by each agent that keeps several TCP connections open at the same time. Some agents may execute on remote machines near the targets in order to improve downloading speed.

Tan's team presents a set of clustered-based policies in their crawler system [19]. Each cluster starts by sampling certain amount of web pages and creates corresponding changing patterns, which is used to predict the change frequency of those web pages. Web pages with higher change frequency usually have higher probability to be re-visited.

Anansi discussed in this paper is a distributed web crawler developed on BOINC and Hadoop DFS. The system utilizes the computing power and resources from voluntary computing. As the same to many other BOINC projects, Anansi has its own project server, which creates and dispatches tasks to voluntary clients. In addition, a cluster of storage servers is controlled by the project server to perform data transition and analysis.

2.3 Voluntary computing and BOINC

Voluntary computing provides a convenient and efficient way to use world's computing power and disk space, which is also known as public-resource computing, global computing or peer to peer computing. Voluntary computing was first introduced by GIMPS and Distributed.net, whose goal is to better utilizing existing computing power from Internet-connected personal computers [1].

The BOINC System can run on many operating system platforms depending on the

requirement of projects. Volunteers can download software BOINC core client corresponding with their OS platform, choose to participate one or more of scientific projects powered by BOINC. A BOINC project has its own server and website. The project uses the website URL as its unique identifier among all BOINC projects. BOINC eases the development of distributed software system by providing programmable interfaces for both server side and client side, along with a network of voluntary participants.

Figure 2.1 shows the infrastructure of BOINC system, which consists of two primary parts: BOINC server and BOINC clients. BOINC server is responsible for creating and sending jobs to BOINC clients. BOINC clients perform computing tasks and return corresponding results to BOINC server. By default, BOINC use first-come-first-serve (FCFS) policy on task scheduling [7]: The BOINC server creates a number of new tasks and dispatches them in the order of creation to involving clients. When sending new tasks to a client, BOINC has no idea about the dependability of the client such as whether they performed previous tasks correctly and punctually. Also there is no policy to handle duplicated new tasks, or to validate results returning by different clients for the same task in BOINC [7].

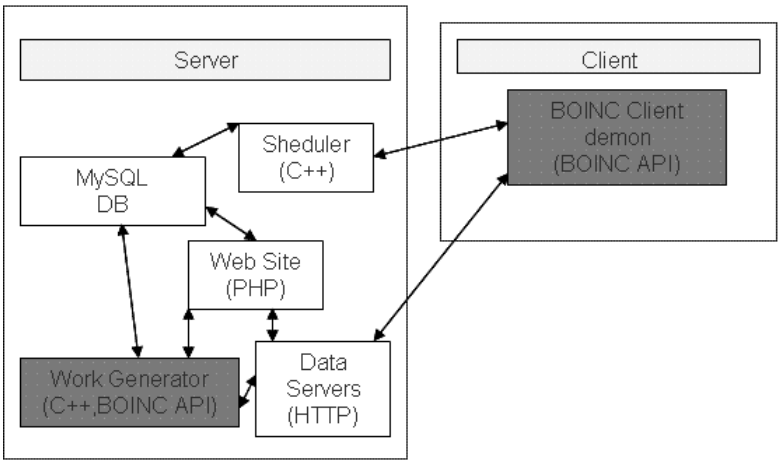

Figure 2.1: The BOINC system from ($http://are.ehibou.com/boinc-boinc-api/$)

2.4 MapReduce and Hadoop

MapReduce was first introduced by Google as a new programming paradigm in its search engine and some other data intensive applications. It is widely used in scientific applications to perform tasks on multiple large data sets. MapReduce provides mechanisms to handle data parallelism and resilience [12]. Since our distributed web crawler has to deal with a large volume of data sets and perform data parallelism, it is ideal to use MapReduce in our implementation.

Hadoop is a software framework that supports data intensive distributed application. It was first inspired by Google's MapReduce and was originally developed to support distribution for the Nutch search engine project. Currently Hadoop is being built and widely used by industry such as IBM, Google and Yahoo to perform their data intensive tasks running on clusters.

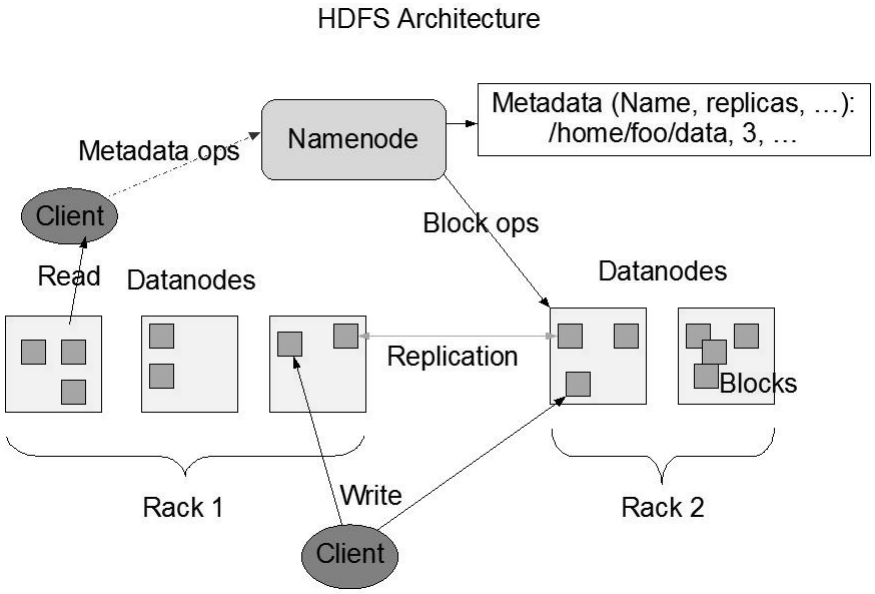

Figure 2.2: HDFS architecture from hadoop official website

Figure 2.2 shows the architecture of Hadoop Distributed File System. The system consists of a name node, an optional secondary name node and a cluster of data nodes. Name node manages dataflow on the file system. The secondary name node and the name

node are used together to create up-to-date directory structure. Data nodes communicate with each other to rebalance data and move copies back and forth. Data nodes transfer data around by block using a protocol specific to Hadoop Distributed File System. MapReduce engine is built on top of Hadoop Distributed File System. It consists of a job tracker and several task trackers. The job tracker has a picture of which node contains the data, and is trying to dispatch tasks to available task trackers closer to the data node. The job tracker is responsible for rescheduling if any of the task trackers fails [21].

Chapter 3

Graphing the web

In this section, we provide definitions for a graphical World Wide Web and discuss its detailed features. We also describe the software architecture of our distributed web crawler.

3.1 Web as a graph

As mentioned above, a web resource in World Wide Web can be identified by a sequence of string characters. A unique string only maps to one web resource on Internet while a web resource can have multiple references at the same time. In our days, Uniform Resource Identifier is the vital technology widely used to present World Wide Web, which is fundamental to many widely used network protocols such as Hyper Text Transfer Protocol (HTTP), Domain Name System (DNS), and TCP/IP.

3.1.1 URI, URL and URN

Uniform Resource Identifier (URI) is also known as a broader class of identifier for two sub identifiers: Uniform Resource Names (URN) and Uniform Resource Locator (URL) [11]. A URI usually consists of a sequence of string characters that indicates a web resources on Internet.

Uniform Resource Names (URN) specifies a web resource by a specific name, which is a sequence of self-explanatory strings separated by colons. A URN identifies the web resource by connecting to a corresponding URL through Domain name system (DNS).

Uniform Resource Locator (URL) is known as the fundamental representation of World Wide Web. Each URL has a domain name that is connected to IP address with Domain Name System (DNS), indicating the Internet address of certain web resource. A URL can be divided into 5 distinct components.

1. **Scheme name**. The scheme name corresponds with certain network protocol. It begins with a letter and terminates with a colon, between which there is any combination of digits, letters, periods, pluses, or hyphens. Scheme name is case-insensitive and specifies the protocol used by the URL.

2. **Authority**. Authority is preceded by a double slash and ends with the next slash. Many URLs contain a hierarchical element for a naming authority that consists of optional user information, a hostname, and an optional combination of a colon followed by port number. User information contains at least a user name with an optional password preceded by a colon. Hostname is presented as a domain name or IP Address which indicates the entry of relevant resources. Default port number defined by the scheme name will be used if it is not specified.

3. **Path**. A path consists of segments in sequential order separated by a forward slash. It specifies the routing of certain web resources located by the URL. Although a path usually looks like clearly specifying the real location certain web resources, there are actually a number of strategies such as redirecting, forwarding that work on exploring the real location of such a resource hidden behind.

4. **Query**. The query is an optional component preceded by a question mark with several segments in the format of $< key >=< value >$ separated either with semicolon or ampersand.

5. **Fragment**. The last optional part called fragment locates at end of the URL. A Fragment starts with an ampersand and specifies the location of a secondary resource. For

example, a subtopic identified by the remainder inside the web page.

3.1.2 Definitions of the graph structure of World Wide Web

Web resources driven by HTTP are present as web pages. World Wide Web can be considered as a directed graph, assuming each URL as an object and any two URL between which one can reach another as a link. To simplify, our study of the web graph focused on mapping nodes only to HTTP web pages that is accessible to general public and contain only text in their contents, in which login web page, HTTP security links and HTTP objects such as media files, images are currently ignored. Following are the features of web graph as we defined.

Nodes. As mentioned above, a URL is a string or IP address that refers to a web resource on Internet which consists of 5 parts: scheme name, authority, path, query and fragment. We consider a URL as valid as long as it has a domain name inside its authority. To simplify, we ignore issues caused by DUST (Different URLs with Similar Text) [3] and identify nodes only by their URLs. According to our assumptions, all URLs on Internet can be identified as nodes in this defined directed graph, whose adjacent nodes either represent a linked URL inside the node's page or a URL that contains the node as its linked URL.

Edges. An edge is also known a link that connects two adjacent nodes inside the graph. The direction of an edge indicates relationship between two connected nodes. The start point of an edge refers to a node that contains the end point node as one of its linked URLs.

In-degree. In-degree is the number of links a node has been referred by. In-degree of a web page indicates its popularity among all websites in the term of being indirectly accessed.

Out-degree. In order to be informative, a web page usually contains one or more related linked URLs inside its contents. The number of linked URLs inside the web page is defined as out-degree of the web page, the value of which varies dramatically in different kinds of websites. Frequently updated web pages such as Amazon.com, CNN.com usually have a higher out-degree value that varies time by time, while personal websites or static web pages may maintain a stable and small value of out-degree for a long time.

3.1.3 Dynamic nature of the web graph

To understand the web graph of World Wide Web, we need to maintain information for each node and edge in the graph and keep them update when necessary. Even though many studies have tried to understand the big picture of the web graph, it is extremely difficult and almost impossible to maintain a up-to-date snapshot for such a graph due to its dynamic nature, which is caused by two primary factors.

The first factor resulting in this dynamic nature is the rapid growth of Internet. Based on recent studies, researchers claim that Internet doubles its amount of registered hostname every 5.32 years [14]. The total number of web pages in those hosts keeps increasing although some of them decrease for many reasons. As the web graph maps each of its nodes to a web page on Internet and each of its edges to a connection between two pages, even a slight change on a single page can result in a great change of the whole graph.

The second factor is the frequent changes happening on Internet. In our days, many popular websites frequently change their contents in order to satisfy existing users and attract more, especially those in e-commerce or media industry such as Amazon.com and CNN.com. The web pages in those websites usually have a large in-degree and out-degree and tend to change frequently and dramatically, which contributes a lot to the dynamic nature of the web graph.

3.2 Crawling a node

The strategy used to crawl a single node in a client is simple: (a) Check existence of the node, (b) Download *robots.txt* from the node and parse it if the node exists, otherwise skip to the next node, (c) Fetch page content of the node, if no contents found from the page, skip to the next node (d) Parse all linked URLs from the contents, excludes links forbidden by *robots.txt*, (e) Transfer relative links into absolute ones, (g) Exclude linked URLs whose *Content-type* is not *text/html*, (h) Fetch HTTP code for each linked URLs.

Each client is running a copy of crawling application independently and crawls the web following the steps above. The crawling application consists of three primary parts: the URL parser, the page downloader and the robots checker.

The URL parser validates the input for each task, which has to be a node. In our system, an input is a node, if it has a domain name or IP address corresponds with a web

Table 3.1: Some of HTTP status code

Status Code	Description
100	Client can continue sending request
200	Successful HTTP requests
302	Temporary redirect is required
404	Requested resource is not found

resource whose HTTP status code is not 4XX or 5XX. Table 3.1 shows some HTTP status codes that are familiar to many web users.

The page downloader is responsible for fetching contents from the web pages. Before fetching a web page, the page downloader sends a HTTP header request to the page host and checks the HTTP status code inside from the response. It only proceeds to the download procedure when the HTTP status code is 200. Figure 3.2 shows an example of a HTTP header response.

Table 3.2: Sample HTTP Header Response

```
Key=Value
null=HTTP/1.1 200 OK
Server=Netscape-Enterprise/4.1
Date=Mon, 11 Feb 2002 09:23:26 GMT
Cache-control=public
Content-type=text/html
Etag="9fa67d2a-58-71-3bbdad3283"
Last-modified=Fri, 05 Oct 2001 12:53:06 GMT
Content-length=115
Accept-ranges=bytes
Connection=close
```

The robots checker enforces robots exclusion protocols on edges found from a node. Many web hosts refine behaviors of cooperating web crawlers by define forbidden paths for certain agents in their *robots.txt*. In our design, only paths that are forbidden from universal agents will be considered exclusive, paths exclusive for other agents such as Google, Baidu bot will be ignored by our crawling application.

3.3 Crawling the web

To crawl the whole web, we utilize a network of voluntary clients to perform crawling tasks respectively and independently. The system originally starts from a few chosen web pages, which are also known as seeds [13].

The crawling application in our system uses re breadth-first search to traverse World Wide Web: the crawling application transverses page contents of an URL, collects directly accessible linked URLs from the page, then starts traversing each of those linked URLs. The system stores and updates the collected information periodically. It also performs data analysis on those data and creates lists of URLs for crawling and re-crawling.

3.4 Collaborative crawling

In our design, the whole system is separated into two distinct components: the scheduling system and the crawling system. The scheduling systems runs on our project servers while crawling system is running on a network of workstations owned by voluntary clients. The scheduling system plays an important role in coordinating the clients. Each voluntary client in the crawling system is running a copy of crawling application independently and working collaboratively with each other.

3.4.1 The scheduling system

The scheduling system consists of a downloader, a uploader, the URL ranking system and the scheduler. The crawling system retrieves new tasks from the downloader and uploads finished ones to the uploader. The scheduler communicates with both the downloader and the uploader, handle requests from clients, and control URL ranking system to perform data analysis.

The URL ranking system performs ranking algorithms on multiple large datasets of URLs. It calculates popularity for each URL collected by crawling system and return corresponding results to scheduler. The scheduler creates new tasks by the order of their popularity in those results. We define in-degree of an URL as number of web pages contains such it and out-degree as number of linked URLs it has inside the page contents. Values of in-degree and out-degree are taken into account for popularity of a URL.

To create new tasks, the scheduler fetches lists of URLs associated with their popularity information from URL ranking system. Each URL become the input of a new task. In BOINC, a task consists of an input file, a workunit template, a result template. In our system, an input file contains an URL that need to be crawled. Workunit template is an unified template for all workunits, it provides a guidance for crawling application to run new tasks. Result template specifies location of uploader for crawling application to report finished tasks. Popularity of a URL is presented by a numeric number, a larger numeric value indicates a higher popularity. As BOINC uses first-come-first-server (FCFS) policy as a primary means for task scheduling, our system inserts new tasks into BOINC server by the priority of the input URL in those tasks.

Each task has been assigned an estimated finished time and responding time when created, both of which were specified in the workunit template. A task is expected to finish within the period of its estimated finished time. Estimated responding time indicates the upper bound of a task's running time, a task running more than this period will be discarded and reported as invalid to scheduler. The scheduler validates results from clients and checks both the client's identity and the content of returning results, it also re-schedule the tasks whose results are invalid.

3.4.2 The crawling system

The crawling system in our design is a distributed system running on a network of voluntary clients, each of which is performing crawling tasks independently using and coordinates with the scheduler.

On the other hand, potential attacks from hackers become possible since the project is public-accessed in World Wide Web. Malicious invasion could result in serious problems such as exposure of users' profile and utilizing involving clients to perform ICMP attack. To prevent users from being manipulated even if the scheduler is broken in, BOINC applies digital signatures on executable files so that the core clients can authenticate them: A pair of public and private keys are generated and the the signature files for executables are created with the private key. When downloading new executables from the scheduler, a client uses both the public-key and the corresponding signature file to validate the executable before running it [20].

For better performance, crawling application maintains local copies for encoun-

tered URL's robots exclusion rule. As robots exclusion protocols in remote hosts are managed autonomously and updated independently of our local copies, crawling application needs to update local copies automatically and periodically.

Chapter 4

Implementation and algorithms

The following describes the detailed implementation and algorithms used in the scheduling system and the crawling system. We discuss the steady solutions as well as encountered issues in each critical part of the system.

4.1 Creating and dispatching tasks

The scheduler is implemented in a BOINC server and creates new tasks with a pre-defined workunit template and an input file. As mentioned above, a task is represented as a workunit in BOINC, and a workunit template defines parameters of the task such as location of the input, estimated finishing time, estimated delay bound and the replication factor of returning results. Daemon *anansi work generator* is responsible for creating tasks and inserted them into the database. It has two threads that are running in the background: one to fetches URLs from the URL ranking system and creates new input files, all of which are dumped into directory *todo*; the other checks directory *todo* periodically and creates new tasks with the input files.

Once a task is created, its input file is copied to directory *download*. The clients communicate with the scheduler periodically, requesting new tasks or reporting finished tasks. Once a connected client is ready to be sent a new task, it requests and download the corresponding input file from *downloader*.

4.2 Algorithm in the crawling system

When a client receives a task, it reads the URL from the input file as well as associative attributes from the workunit template. After parsing and validating the input URL, the crawling application in the client starts out crawling immediately and returns results to the server when it finishes. We use Perl Compatible Regular Expression (PCRE) [4] to parse and validate URLs in the crawling application, which is a Regular expression library for C and C++.

An input will be considered as valid only if it is a URL with a domain name. Following are the steps to parse and validate an URL: the crawling application assumes an input is a single URL and try to split it into 5 parts. If no domain name is found in the URL's authority or its scheme name is specified other than Hyper Text Transfer Protocol (HTTP), the crawling application will abort the task and report to the scheduler. BOINC server then tries to schedule another new task to the client. An URL with no scheme name is appended with *http* by default. Fragments are also removed from the URL.

When a URL has been parsed, the crawling application sends a HTTP header request to the URL server. The HTTP header response contains a HTTP status code, which specifies current status of the resource. The crawling application will go to the next procedure only if the status code is 200, which indicates the existence and accessibility of a web resource. Any other status code will cause abortion the task.

The crawling application then fetches robots exclusion rules from *robots.txt* that locates in the URL server. We assume absence of *robots.txt* indicates the permission to crawl all contents in such a URL server. To reduce network traffic and memory I/O, crawling application uses a hash structure to store contents inside *robots.txt*. The hash structure uses the URL as the key and a linked list that stores contents in *robots.txt* as its value. Each element in the linked list corresponds with the pre-compiled PCRE pattern of a unique line inside *robots.txt*. Once the *robots.txt* file is downloaded successfully, the crawling application proceeds to fetch the page contents. If no page content is found, the crawling application would skip to the next URL and report it to the scheduler later. The crawling application then extracts all linked URLs from the page and pushes them into a buffer. In our system, we only extract linked URLs from attribute *href* inside tag **a**.

The crawling application then remove linked URLs whose scheme name is not

HTTP from the buffer. For the remainders, the crawling application removes fragments for each of them and transfer relative ones into absolute linked URLs. The last and most time consuming step is to send HTTP header request to each URL left in the buffer, the response of which contains HTTP status code, content-type and other HTTP attributes. Linked URLs with HTTP status code other than 200 or 302 or content-type other than *text/html* are removed from the buffer. Until now the crawling application finishes crawling this URL and sends the output back to the scheduler.

4.3 Policies in the scheduling system

The scheduling system consists of the scheduler and the URL ranking system. The scheduler manages dataflow for the whole system, which is responsible for communicating with the crawling system and the URL ranking system. The scheduler also serves as a task downloader and a result uploader. The URL ranking system stores and prioritizes URLs collected by the crawling system.

4.3.1 Validating results from clients

Validating returning results from clients is necessary to prevent users from returning invalid results to the server on malicious purpose. The scheduler must check both the client's identity and the content of returning results when processing validation. BOINC originally provides a mechanism for validation: the server sends additional copies of workunit to different users and then compares results returning from them. This solution is useful for computation that returns exactly the same results for certain inputs such as analysis of a graph pattern or signal but is not suitable for those whose computation highly rely on timestamp.

As mentioned above, the rapid growth of World Wide Web associates with dynamic changing in both the content of web pages as well as their connected pages, the crawling results with different starting timestamps for the same URL varies dynamically. Moreover, the crawling results returning from different users but with the same starting timestamp may also vary due to the difference on bandwidth and computing abilities between those users. In this case, validating results by sending additional copies of workunits to different users are not reliable for our system.

In our system, the scheduler only sends one copy of a workunit to one user. The crawling application in the client side encrypts the result by appending additional symbols to its end before reporting. Whenever the scheduler receives results from the clients, it decrypted the file by checking additional symbols in the content. Existence of the appended symbols indicates the results are valid.

4.3.2 Storing nodes using multi-level directories

One efficient way to search and update information of collected nodes is to store them using the structure of multi-level directories. To achieve this, we have to make sure the path of the multi-level directories is unique for each distinct node, which requires the combination of those directories' names for each node is unique. One possible solution is to encrypt each node into a unique sequence of hashed character and separate that into smaller pieces in sequential order, each of which is used to name the different level of directory.

The more level directories a node is represented by, the more efficient for it to be found. For example, assuming we have collected **n** nodes and the scheduler stores nodes using a one level directory structure, in which case **n** directories are created and each of them stores only information for a single node. The complexity to search for a node among all those directories is O(n), no matter what kind of hash algorithm is being used. In another case, if the scheduler uses two-level directories to store a node, and the hashed value for a single node has a length of **m** characters, the first **k** ($k <= m$) of which are used to represent the first level directory, then the complexity to search for a node is O($\frac{n}{2^k}$), which is definitely less than O(n) on average.

Currently in our system, the scheduler stores all collected nodes using a two-level directory structure. When it receives a node that has been craweled by the clients, it first hashes the node's URL into a 40 character long hexadecimal value using the SHA-1 algorithm. The first 2 characters of the hexadecimal value are used as the name for the first level directory and the remaining 38 characters are used as the name for the secondary level directory. Information such as the collected linked URLs, node's original URL and the timestamp are stored in separate files under the secondary level directory. As mentioned above, the crawling system returns only nodes whose HTTP status code is 200 or 302 back to the scheduler. Currently in our solution, the scheduler stores and maintains information of all those nodes as long as their HTTP status code was 200 or 302 when they were first

found.

Information of a node is stored in separate files under its own directory, which includes the node's URL, its timestamp for the latest visit and the links found in the latest visit. The value of timestamp and contents of links are updated whenever the node was re-visited. Both the change ratio in links between two consecutive visits and age of a node can be calculated when it was re-visited.

4.3.3 Prioritizing nodes

As mentioned before, the URL ranking system prioritizes collected links and creates lists of URLs for crawling and re-crawling. It prioritizes new nodes and old nodes (nodes that were visited before) respectively by applying different strategies. Following are parameters taken into account for prioritizing the nodes.

In-degree: number of nodes that refer to the current node. In-degree of a node intends to change as the number of crawled nodes increases over time. After enough data has been collected, the value of in-degree somehow indicates the popularity of a node among all URLs in World Wide Web, which may vary from zero for new nodes to hundreds for the popular ones. In-degree of a node is mainly determined by other nodes in Internet but may have something to do with the contents of the node.

Out-degree: number of linked URLs a node contains. Value of out-degree is determined by the node itself. A node can easily changes its out-degree value by increasing or decreasing number of linked URLs it contains.

Age: The value of age is represented by ratio **T** and is determined by the difference in timestamps between the latest visit and the current timestamp. The value of a timestamp is the seconds that have elapsed since January 1st 1970. The value of **T** is an extremely large number as it is represented in seconds. In order to balance **T** and other parameters, we reduce its value by dividing it by the estimated seconds to crawl the whole web T_{crawl}, which guarantees that the value of textbfT correctly reflect its priority for re-visited during the whole crawling process. Equation 4.1 shows the strategy mentioned above. In the equation, T_{latest} is the timestamp for the latest visit and T_{now} is the current timestamp. T_{crawl} is the length of the estimated time to crawl the whole web, which is currently defined as 6 months in our system.

$$T = \frac{T_{now} - T_{latest}}{T_{crawl}} \tag{4.1}$$

Change: Change **C** is the ratio of changed links between two continuous visits for the same node, which can be represented by the set of links found in latest visit E_{latest} and the set of links found in the previous visit E_{prev} in Equation 4.2. The value of **C** is between 0 and 1. If both E_{prev} and E_{latest} are empty, **C** is set to 1.

$$C = 1 - \frac{\mid E_{prev} \cap E_{latest} \mid}{\mid E_{prev} \cup E_{latest} \mid} \tag{4.2}$$

The URL ranking system uses different strategies to prioritize new nodes and old nodes and store them in two different lists. Priority of a new node is determined only by its in-degree that indicates its popularity. The reason for this strategy is simple: crawling a popular node has a higher possibility to find the other popular ones. Equation 4.3 shows the relationship between the priority **P** of a new node and its value of in-degree **I**. A new node with higher priority has a larger value of in-degree and versa vice.

$$P_{new} = I \tag{4.3}$$

To prioritize the list of old nodes for re-crawling, the URL ranking system needs to consider the values of in-degree **I**, out-degree **O**, age **T** and contents difference **C**. The priority of an old node can be represented by Equation 4.4.

$$P_{old} = CT(I + O) \tag{4.4}$$

The strategy to prioritize old nodes is simple and straightforward: a node having elder age has higher priority as it needs to be kept up-to-date; a node having larger ratio of change in links has higher priority; a node having higher in-degree and out-degree has higher priority as that indicate its popularity. According to Equation 4.4, an old node with a higher value on anyone of **C**, **T**, **I** or **O** has a higher priority than another when the other parameters are identical, which satisfies our expectation on prioritizing the URLs.

4.4 System scaling

The benefits of building our system on distributed infrastructure is its scaling capability. The performance of the system can be scaled up easily by adding low-cost workstations. On the server side, scheduling system can speed up computing in MapReduce program in Hadoop DFS by adding additional servers in the HDFS cluster. So far Anansi server is also running as a downloader and result uploader, as long as scheduler for the whole system. One solution to scale up the system is adding more task downloaders and result uploader.

We estimate a task downloader can handle around 5 users in one second, while a result uploader can serve a similar number of users. An increase number of participants on our BOINC project facilitate the growth of crawled URLs. he bottleneck here is the computing capabilities of involving voluntary machines, as well as the working schedule of them. Currently there are more than 300 users involving in the project, each of whose machine usually vary dramatically to others on their computing capability or network bandwidth.

Chapter 5

Results and analysis

In this section, we present relevant data of our crawled results and discuss the feasibility and practicability of using the system to crawl the whole web. The data come from the experiment we conducted between September 2009 and November 2009, in which the number of active users are increasing rapidly from 10 at the very beginning to more than 100 at the end of November 2009. The system crawled around 400,000 nodes and found about 80,000,000 linked URLs from those nodes around two months.

5.1 Experimental results

Figure 5.1 shows the total amount of crawled nodes from the beginning to certain days. As mentioned above, in order to perform crawling and re-crawling at the same time, 90 percent of the workunits dispatched to clients are created with new nodes while the other 10 percent are created with crawled nodes. From Figure 5.1, we can tell that the system is working correctly under this rule: both the number of total crawled nodes and unique nodes are increasing and the ratio between them are around 10:9 for the whole period. We can also tell that the absolute crawling speed of the system keeps increasing as the slope has been rising so far. Currently the system could crawl around 20,000 nodes (including obtain http code for each linked URLs found in those nodes) with the help from 100 active users.

Figure 5.2 shows the number of nodes with the same value of out-degree and Table 5.1 shows the average number of out-degree values from the crawled nodes. In Figure 5.2,

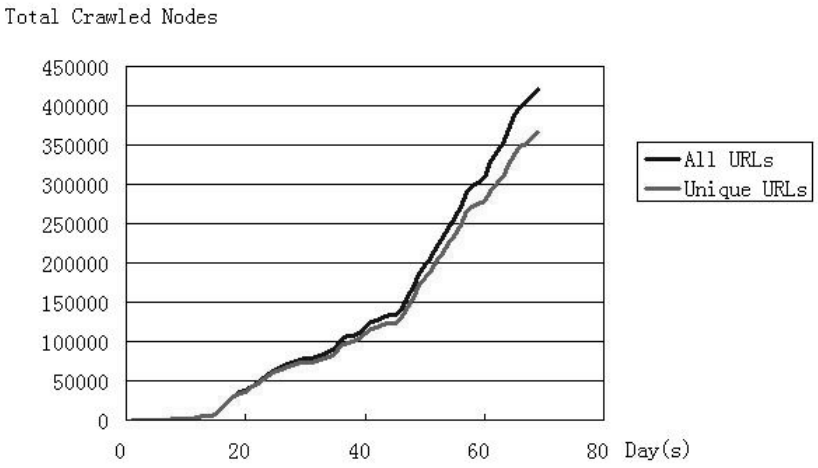

Figure 5.1: Total URLs crawled by days

the medium number of crawled nodes is 20956 when out-degree equals to 202, which is closed to the average out-degree value 205.322 shown in Table 5.1.

Table 5.1: Average out-degree from crawled nodes

number of edges	number of nodes	Avereage out-degree
86,438,305	420,989	205.322

We use **I** for value of in-degree and **O** for that of out-degree. Table 5.2 shows performance of our URL ranking system built on Hadoop HDFS system with a number of 3 datanodes. The data shows that the average processing speed is related to the value of $\frac{I}{O}$.

5.2 Estimations on crawling the whole web

The system schedules re-crawling on encountered nodes as well as crawling on new found nodes. Assuming crawling speed of the system is **F** nodes per time unit, α ratio of which are new nodes, then the number of new nodes crawled in a period of **t** can be represented as **n** in Equation 5.1. The value of α directly determines the total CPU time spent on crawling new nodes while value of $1-\alpha$ determines that on re-crawling. Currently

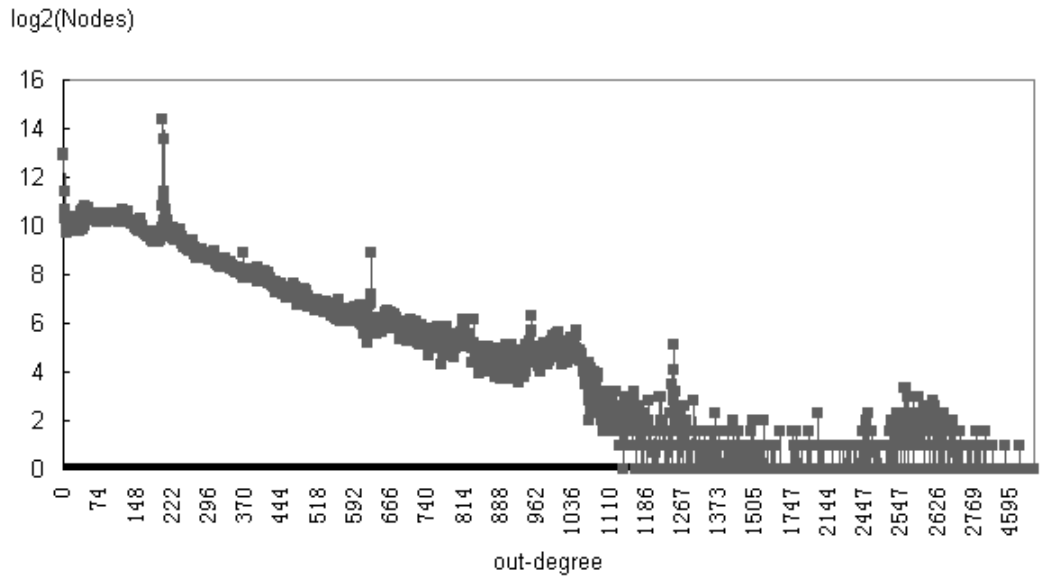

Figure 5.2: Summary of crawled nodes and their out-degree values

the system uses 0.9 as the default value of α.

$$n = \alpha F t \tag{5.1}$$

To estimate the time to crawl the whole web, we first assume the total number of nodes in World Wide Web is **N**, the average out-degree for each page is **S**, and the crawling speed of our system is F nodes per time unit. So the time units needed to crawl the whole web is t=$\frac{N}{\alpha F}$.

According to our assumption, World Wide Web contains a total number of N nodes, whenever a number of nodes have been crawled, the linked URLs found from those nodes will become part of the pending list for crawling tasks. Assuming the average out-degree of a node is S, which is closed to the average number of found lined URLs in a node according to Table 5.1.

To simplify, we assume the system only crawls new nodes ($\alpha = 1$) and use P(n) to represent the size of pending list after **n** nodes have been crawled. Whenever **(n-1)** nodes have been crawled, the pending size is P(n-1), the probability of crawling a new node next time is $\frac{N-(P(n-1)+n)-1}{N}$, after which the total size of pending size is P(n). Equation 5.2

Table 5.2: Performance of the URL ranking system

input	output	compression rate	duration	processing speed
500,000	323,963	1.543386	33	15,151
1,000,000	582,692	1.716173	58	17,241
1,500,000	801,783	1.87083	79	18,987
2,000,000	1,000,667	1.998667	103	19,417
2,500,000	1,199,649	2.083943	94	26,595
3,000,000	1,374,316	2.182904	111	27,027
3,500,000	1,542,274	2.269376	120	29,166
4,000,000	1,710,902	2.337948	138	28,985

shows the relationship between P(n) and P(n-1) based on our assumption, which can be simplified as Equation 5.3.

$$P(n) = P(n-1) + S[\frac{N - (P(n-1) + n) - 1}{N}] - 1 \tag{5.2}$$

$$P(n) = (1 - \frac{S}{N})^{n-1}(S + 1 - N) + (N - n) \tag{5.3}$$

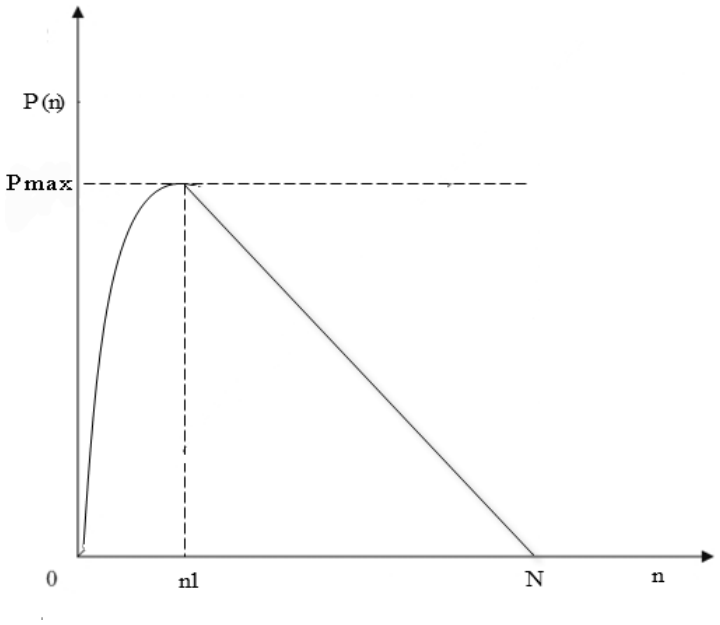

Figure 5.3: Relatinships between size of pending list and crawled nodes

Taking derivative with respect to **n** yields Equation 5.4. According to th abstract graph P(n) in Figure 5.3, the size of pending list ($P(n)$) keeps increasing until it reaches the maximum value when $D(n)$ equals 0 (point n1), in which **n1** satisfies the equation 5.5 and P_{max} is the maximum value of P(n). After that, the size of pending list starts decreasing linearly and become zero when $n = N$, which indicates the completion in crawling the whole web.

$$D(n) = (n-1)(1-\frac{S}{N})^{n-2}(S+1-N)-1 \tag{5.4}$$

$$(n-1)(1-\frac{S}{N})^{n-2}(S+1-N)-1=0 \tag{5.5}$$

Table 5.3 shows the maximum value of pending list P_{max} and its corresponding value of crawled nodes n_1 when **N** is different and **S** is equal to 200. According to the table, the larger **N** is, the smaller the ratio of $\frac{n_1}{N}$ is, based on which and Equation 5.1, we can claim that if both the crawling speed **F** and **N** increase at a same ratio, the total time to crawl the whole web will decrease as the ratio gets larger.

Table 5.3: The values of P_{max} with corresponding n_1 based on different N

N	P_{max}	n_1
1,000,000,000	968,508,416	26,491,585
2,000,000,000	1,965,042,680	29,957,321
3,000,000,000	2,963,015,354	31,984,648
4,000,000,000	3,961,576,944	33,423,058
5,000,000,000	4,960,461,226	34,538,777

5.3 Feasibility study of the system

In this context we discuss feasibility of the system, which is tractable and scalable. We predict size of the Internet based on a conceivable model as well as present strategies used to collect, store and analysis the encountered data.

5.3.1 Size of the internet

As mentioned above, the size of internet keeps growing in an extreme rate as the number of web users are increasing. In this case, assessing the size of the internet is a difficult proposition due to its distributed body and changing contents. When estimating how large the internet is, some simple metrics such as how many websites in the internet and how many bytes of the total data can usually provide a straightforward and conceivable view. However, even the biggest company who dedicating on exploiting the internet could not answer this question confidently. According to Eric Schmidt, the CEO of Google, who owns the worlds largest index of the Internet, estimated World Wide Web's size at roughly 5 million terabytes of data. Schmidt also mentioned that Google has only indexed roughly 200 terabytes of the estimated data, which is less than 1 percent of the total size, in its seven years of operations from 1998 to 2005 [8].

Some researchers believe that the prediction model established by Moore's Law can also be applied the Internet, in which the metric on number of domain names determines the growing size of the web [14]. Based on the data collected between December 2001 and December 2006, researchers claim that the Internet doubles its size every 5.32 years. According to their model, the amount of registered hostname will reaches around 200,000,000 in the end of 2011 [9].

To simply, we assume the average in-degree for each node is 10, as the average value of out-degree is around 200 according to Table 5.1, we can estimate that the Internet will contain around $\mathbf{N}= \frac{200{,}000{,}000\times 200}{10}$ =4,000,000,000 nodes (web pages) in the end of 2011.

5.3.2 Crawling the web within a period

One of the most time-consuming processes to understand the web graph is to collect sufficient information in a limited time, which is the responsibility of the crawling system in our design. Each client in the crawling system performs crawling tasks independently while cooperating with the scheduler server on receiving new task and report finished results. The crawling application running in a single client fetches a page in the following steps: (a) Fetch the page content into buffer, (b) parse all linked URLs from the content, (c) Dump URLs that is forbidden by robots exclusion, (d) Fetch http code for all remaining linked URLs. Both step (b) and (c) takes a constant and trivial CPU time in general workstations which

can be ignored while step (a) and (c) may vary dramatically depending on the bandwidth and network traffics.

According to Equation 5.1, the time to crawl the whole web is $\mathbf{t}= \frac{N}{\alpha F}$ $(n = N)$. For example, if we want to traverse the whole web without revisit ($\alpha = 1$ and $n = N$) in 6 months (15,552,000 seconds), we need to crawl at least $\mathrm{F}=\frac{N}{t} = \frac{4{,}000{,}000{,}000}{15{,}552{,}000} = 257$ per seconds, with the assumption that the number of nodes is a constant during the whole period of crawling.

Assuming all voluntary participants are connected to a high-performance network and are free of network traffic when processing the crawling tasks, in which case fetching http code from a single page takes 0.1 second. Based on the statistics above, the average out-degree value for each node (page) is 200, so the average time to crawl a node in a single workstation is 20 seconds, which is equal to 0.05 node per second. In this case, in order to crawl the whole web in 6 months, the number of active participants we need is $\mathbf{p}= \frac{F}{0.05} = \frac{257}{0.05} = 5,140$, which is tractable and possible since there are more than 1,733,933,741 web users according to the latest statistics on Internet population in 2009 [18].

5.3.3 Estimations of data storage, bandwidth and data analysis

To maintain the changing status of web graph, the system server stores all data collected from voluntary clients and keep updating them periodically. Those data includes crawled nodes and encountered edges associated with their timestamps. Based on the assumption that the Internet contains around $\mathbf{N}= 4,000,000,000$ nodes, the total space to store those nodes is determined by the average size of their string representations. As mentioned above, the URL string used to represent a node or an edge contains an average number of around 100 characters, which takes around 100 bytes of harddisk space in the system server. However, as we implenent the URL ranking system using Hadoop HDFS filesystem, which stores file in multiple machines and achieves reliability by replicating the data with the default replication factor 3, we actually need 300 bytes to store a single URL [21]. Based on the assumptions above, we can now estimate the total space needed to store those data. To simlify, we ignore size of associated information such as timestamp and changed status, so the total size of required storage is $\mathbf{S}= 4,000,000,000 \times 300 bytes = 1,200$ Gigabytes.

To estimate total consumed bandwidth during the whole crawling process, we have to know the total size of data transmitted between the system server and clients. As mentioned above, a workunit in BOINC consists of an input, a workunit template, a result template and one or more results. The server and clients are actually exchanging infomration on workunits when they are communicating with each other.

In our system, a workunit corresponds with a node and contains only one result, the size of workunit template and result template is fixed and is around 600 bytes for each workunit. In this case, the size of data that clients download from the server is the size of World Wide Web **N** plus the size of template files and associated TCP/IP headers. Equation 5.6 shows the formula to calculate the downloading bandwidth in the server, we assume the average size of TCP/IP headers is 30 bytes in a task and the average size of a URL is 100 bytes. According to the equation, in order to crawl the whole web in 6 months (t=15,552,000 seconds), the downloading bandwidth is around $B_{download}$= 0.183 Megabytes/s.

$$B_{download} = \frac{N \times 600bytes + N \times 100bytes + N \times 30bytes}{t} \tag{5.6}$$

To calculate the uploading bandwidth on the server side, we need to estimate the total size of data uploaded from clients. Again assuming each URL takes 100 bytes and the TCP/IP header is of 30 bytes, as the average out-degree value **S** is 200, the size of result in each task is around $100bytes \times 200 = 20,000bytes$. Equation 5.7 shows the formula to calculate the uploading bandwidth on server, in which when **t** is equal to 6 months (15,552,000 seconds), the uploading bandwidth is B_{upload}= 5 Megabytes/s.

$$B_{upload} = \frac{N \times 20,000bytes + N \times 30bytes}{t} \tag{5.7}$$

We can further reduce the downloading and uploading bandwidth by compressing the data before transferring and uncompress them after receiving. Currently all data being transferred between server and clients are in the format of text files, which can be efficiently compressed and fully recover. The compression rates between the original and zipped files are also pretty high, which is on average 10:1. Based on this assumption, the downloading and uploading bandwidth after compression becomes $B_{downloadCompress}$= $\frac{B_{download}}{10}$=0.0183 Megabytes/s and $B_{uploadCompress}$= $\frac{B_{upload}}{10}$ = 0.5 Megabytes/s, respectively.

Finally, analysis on collected data plays the most important part in assuring correctness of the system. As mentioned above, priority of URLs is determined by parameters such as timestamps, in-degree, out-degree and changed contents which indicate their popularity in World Wide Web. In our system, the URL ranking system performs analysis on collected data, whose output is a pending list of URLs ready for crawling. The system then crawls the whole web by travering URLs from high priorities to lower ones. Assuming the system crawls the web without revisit any encountered URLs ($\alpha = 1$), according to Euqation 5.3 and Figure 5.3, the size of pending list reaches its peak when number of crawled nodes **n** satisfies Equation 5.4, which is the maximum number of URLs that our URL ranking system needs to handle during the whole process and is pretty closed to number of total nodes in the Internet. To rank all URLs in the Internet, assuming the URL ranking system takes **N**$= 4,000,000,000$ input records. It requires **S**$= 1,200$ Gigabytes to store those records and the number of output records is equals to that of the input, which indicates the compression rate is 1 between the input and output. To simplify, we assume the duration to rank all URLs is only determined by amount of input records and number of Hadoop HDFS datanodes. Also according to Table 5.2, the lower the compression rate is, the slower the processing speed is, by which we can assume the processing speed **V**$= 10,000$ nodes per second when compression rate is 1 in our case. As a result, the estimated duration to rank all URLs using 3 datanodes is **T**$= \frac{N}{V} = \frac{4,000,000,000}{10,000} = 400,000 seconds$, which is only 0.2 percents of the total time required to crawl the whole web.

Chapter 6

Conclusions and Future Work

In this paper we presented an open solution to discover the graph structure of World Wide Web using voluntary computing library BOINC and Hadoop MapReduce framework. We also presented our experimental results as well as a feasibility study on crawling the web using our solution. We specially focused on scheduling strategies on server side and crawling policies in clients. Obviously there are many improvements can be made in the system. Two major issues for future work are as below.

1. A detailed study of a more efficient re-crawling policy. This issue can be divided into two parts: On the client side, we are looking for a better crawling mechanism. A possible solution is to collect more useful information such as details in a URL's HTTP header as well as contents of its web page. A multi-threaded solution may also help to improve the performance of crawling. On the server side, a more efficient and reasonable ranking algorithm is needed. To address this issue, we need to collect sufficient information from a large amount of URLs which can provide a good reference.

2. A better validating mechanism on crawling results from different clients. Since many website we encountered are likely to change frequently, especially commercial sites like Yahoo, Amazon.com and CNN.com. This brings difficulty in validating two crawling results for the same node from different clients if they differ greatly in timestamps. One possible solution is to divide timestamp into small slots and only validates crawling results falling at the same slots for the same node, which needs further study to

find a reasonable value of the slot.

Our main interest is in using this distributed web crawler in our research group to look at more other knowledge in Internet. It is possible to build a search engine from our system and uncover more knowledge of the current web graph as more modules are added and more information is collected.

Bibliography

[1] David P. Anderson. Boinc: A system for public-resource computing and storage. pages 4–10, 2004.

[2] David P. Anderson, Carl Christensen, and Bruce Allen. Designing a runtime system for volunteer computing. *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 126, 2006.

[3] Ziv Bar-Yossef, Idit Keidar, and Uri Schonfeld. Do not crawl in the dust: Different urls with similar text. *ACM Trans. Web*, 3(1):1–31, 2009.

[4] Michela Becchi and Patrick Crowley. Extending finite automata to efficiently match perl-compatible regular expressions. pages 1–12, 2008.

[5] Michael Brinkmeier. Pagerank revisited. *ACM Trans. Internet Technol.*, 6(3):282–301, 2006.

[6] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. pages 124–135, 2002.

[7] D. A. Taufer M. Teller P. J. Kerstens A. Anderson D.P. Estrada, T. Flores. The effectiveness of threshold-based scheduling policies in boinc projects. *e-Science and Grid Computing, 2006. e-Science '06. Second IEEE International Conference*, pages 88–88, 2006.

[8] Wise geek. How big is the internet. http://www.wisegeek.com/how-big-is-the-internet.htm.

[9] Qing-Feng Yang Su-Qi Cheng Guo-Qing Zhang1, Guo-Qiang Zhang and Tao Zhou. Evolution of the internet and its cores. *New Journal of Physics*.

[10] Gueorgi Kossinets, Jon Kleinberg, and Duncan Watts. The structure of information pathways in a social communication network. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 435–443, New York, NY, USA, 2008. ACM.

[11] Kisung Lee, Jin Hyun Son, Gun-Woo Kim, and Myoung Ho Kim. Web document compaction by compressing uri references in rdf and owl data. pages 163–168, 2008.

[12] S. Bent J. Lopez-J. Habib S. Wang J. Mackey, G. Sehrish. Introducing map-reduce to high end computing. *Petascale Data Storage Workshop, 2008. PDSW '08. 3rd*, pages 1–6, 2008.

[13] Shan Lin. You meng Li. Qing-cheng Li. Information mining system design and implementation based on web crawler. *System of Systems Engineering, 2008. SoSE '08. IEEE International Conference*, pages 1–5, 2008.

[14] E Mollick. Establishing moore's law. *Annals of the History of Computing, IEEE*, pages 62–75, 2006.

[15] S. Sharifi M. Nasri, M. Shariati. Availability and accuracy of distributed web crawlers: A model-based evaluation. *Computer Modeling and Simulation, 2008. EMS '08. Second UKSIM European Symposium*, pages 453–458, 2008.

[16] M. Ángeles Serrano, Ana Maguitman, Marián Bogu ná, Santo Fortunato, and Alessandro Vespignani. Decoding the structure of the www: A comparative analysis of web crawls. *ACM Trans. Web*, 1(2):10, 2007.

[17] T. Shkapenyuk, V. Suel. Design and implementation of a high-performance distributed web crawler. *Data Engineering, 2002. Proceedings. 18th International Conference*, pages 357–368, 2002.

[18] Internet World Stats. World internet users and population stats. http://www.internetworldstats.com/stats.htm.

[19] Qingzhao Tan, Prasenjit Mitra, and C. Lee Giles. Designing clustering-based web crawling policies for search engine crawlers. pages 535–544, 2007.

[20] Wikipedia. Codesigning-boinc. http://boinc.berkeley.edu/trac/wiki/CodeSigning.

[21] Wikipedia. Hadoop. http://en.wikipedia.org/wiki/Hadoop.