LOCATION AWARE WEB-CRAWLING WITH THE USE OF MIGRATING CRAWLERS

Odysseas Papapetrou

December 22, 2003

Contents

Summary

Most of the modern search engines are based in the well-known web crawling model. A centralized crawler or a farm of parallel crawlers are dedicated to the process of downloading the web pages and updating the database. However, the model fails to keep pace with the size of the web, and the frequency of changes in the web documents. For this reason, there have been some recent proposals for distributed web crawling, trying to alleviate the bottlenecks in the search engines and scale with the web.

In past work [36, 35] we proposed a distributed web crawling paradigm based on mobile agents, called UCYMicra. We also evaluated the model, which proved to be significantly faster and more efficient than the traditional, centralized approach. Our proposal, powered from a paradigm especially developed for mobile code (the mobile agents paradigm), did not have the inefficiencies and concerns of earlier distributed crawling alternatives. Furthermore, some important novel optimizations applied in UCYMicra were helping the system outperform all its opponents. However, neither UCYMicra, none of the other distributed crawling proposals was considering a location aware model for faster downloading of the web pages.

In this work, facilitating the flexibility and extensibility of UCYMicra, we consider adding location awareness to distributed web crawling, so that each web page is crawled from the web crawler logically most near to it, the crawler that could download it the faster. We evaluate the location aware approach and show that it can reduce the time demanded for downloading to one order of magnitude. We also propose two different heuristics to succeed location awareness in a distributed web crawling system with negligible overhead. The evaluation of these heuristics reveals that the distributed location aware approach can improve distributed web crawling dramatically without important overhead, and can help toward alleviating the bottlenecks faced from the commercial search engines.

Chapter 1

Introduction

1.1 General

Internet is considered from many scientists the great revolution of the last decade. The most known application running over the Internet, World Wide Web, managed to get into every scientific laboratory and every school of the modern world, and become one of the most essential tools for knowledge, teaching, researching, even commerce.

World Wide Web is expected to contain several billions of available documents, and from many researchers is seen as a big (the biggest) database system. However, the organization of documents in the WWW does not facilitate easy searching or locating data. WWW, unlike any other database system, does not include any indexing system itself, that can help discovery of data easy and efficient. Instead, the millions of users of the WWW are solely dependent on search engines for searching through the Internet.

The task of updating the information stored in the search engine databases is usually performed by web crawling. A web crawler, or most usually, a farm of web crawlers are assigned the task of downloading pages from the web, processing them, and integrating the processed results in the search engine database. Web crawling however is limited from several severe bottlenecks. For this reason we recently proposed a distributed web crawling approach able to solve these bottlenecks and scale gracefully with the Internet. UCYMicra was powered from mobile agents which were migrating in the available hosts and performing web crawling of the pages assigned to them in low use hours.

The new approach presented in this work is built in the UCYMicra application.

More specifically, UCYMicra is now enhanced with location awareness. We propose the delegation of each URL to its most near available web crawler in order to reduce the required time to download the web pages. We suggest two novel ways to enable the location-based delegation, that run with negligible network and processing overhead and have near-to-optimal results.

1.2 Motivation

Indexing the Web has become a challenge due to the Web's growing and dynamic nature. A study released in late 2000 reveals that the static and publicly available Web (also mentioned as surface web) exceeds 2.5 billion documents, while the deep Web (dynamically generated documents, Intranet pages, web-connected databases etc) is estimated to be three orders of magnitude larger [30]. Furthermore, Google [22] now reports more than 3.3 billion documents crawled. Other studies show that the Web is growing and changing rapidly (more than 40% of the pages per week) [9], while no search engine succeeds coverage of more than 16% of the estimated Web size [28].

Web crawling (or traditional crawling) has been the dominant practice for Web indexing by popular search engines and research organizations since 1993, but despite the vast computational and network resources thrown into it, traditional crawling is no longer able to catch up with the dynamic Web. Consequently, popular search engines require up to 6 weeks in order to complete a crawling cycle for the non-paying sites [44]. Moreover, the centralized gathering nature of traditional crawlers and the lack of cooperation between major commercial search engines are two more reasons toward these inadequacies.

The absence of a scalable crawling method triggered some significant research in the past few years. For example, Focused Crawling [5] was proposed as an alternative method but it did not introduce any architectural innovations since it relied on the same centralized practices of traditional crawling. As a first attempt to improve the centralized nature of traditional crawling, a number of distributed methods have been proposed (Harvest [3, 2], Grub [29]), yet, with important limitations. Fielder and Hammer in [18, 15] also proposed a distributed crawling method. However, all of these proposals had important limitations prohibiting them from being utilized in the actual Internet environment.

Trying to solve the problems of centralized crawling as well as the limitations of distributed crawling proposed in [29, 18, 15, 3, 2], we recently suggested UCYMicra [36], a distributed crawling methodology powered by mobile agents, migrating to different hosts in the Internet, crawling the web and sending back the processed data. However, while the initial proposal of UCYMicra outperformed traditional crawling significantly,

it did not take full advantage of the offered distribution. Any new URL was delegated randomly to one of the available migrating crawlers, taking no consideration for the location of the URL.

For this reason, in this work we suggest applying location awareness in UCYMicra, which will enable near-optimal delegations of the new URLs to the migrating crawlers and eventually reduce the downloading time. We propose two distinct approaches toward location awareness. Both the approaches have impressive results with negligible execution overhead.

1.3 An overview

Following this brief introduction, we will describe the mobile agents paradigm, which is used for the implementation of this work. In chapter 3 we will discuss the architecture of a complete search engine, look into the three basic parts of a search engine, and mention the current trends in keeping the search engine's database updated. Then, we will briefly describe UCYMicra, a previous work that will be used in this work.

Following, in chapter 5 we will propose and describe location aware web crawling, and show how this can help toward alleviating important bottlenecks from the web crawlers. Then, we will propose and evaluate HiMicra and IPMicra, two methods that can actually perform a location aware delegation of domain names to the available URLs, without expensive probing. In chapter 8 we will report on the evaluation of location aware web crawling, HiMicra, and IPMicra. Finally, we will conclude with our final conclusions from the project and some planned future work.

1.4 Tools and Languages

The implementation of this project requires Sun's Java 2 Platform, Standard Edition (Java 1.4 or newer). We also require Voyager mobile agents platform, available from Objectspace. For database storage, we use the MySQL database server, and for the connection of the software with the database we use JDBC drivers. All the required software is already installed in the department's servers.

Chapter 2

An introduction to the Mobile Agents paradigm

2.1 Introduction

In this chapter we will look into the mobile agents paradigm which is used during the implementation of this work. We will start by describing the mobile agents paradigm. Then we will demonstrate the main application areas of the paradigm.

2.2 Mobile agents: A paradigm for the future

Mobile agents are autonomous processes, dispatched from a source computer to a target host to accomplish a specific task [6]. The mobile agents include the computation (the code) along with their data (variables) and execution state. When a mobile agent decides to dispatch to a remote server, it ceases working and moves to the target host over the network (as a serialized object). As soon as the agent code arrives to the target server it is received from the mobile agents platform and reassembled to the original mobile agent instance. Then the mobile agent continues working.

The mobile agents paradigm was initially proposed as a distributed computing paradigm. However, the flexibility of the model soon enabled many new applications, mainly in the area of Internet and wireless (mobile) computing. Many distributed computing applications use mobile agents to remain operational during network disconnections. There are also many suggestions for using mobile agents for more efficient facilitation of the network resources.

The advantages of the mobile agents paradigm are very important. The new paradigm even enables applications that could not be implemented at all without it. More specifically, the mobile agents paradigm has the following advantages:

Simplifies distributed applications The mobile agents model simplifies the implementation of distributed systems based on asynchronous communication [16, 6, 17] (even very large systems that would be difficult to implement with any other approach). Furthermore, due to the independence of the mobile agents from the underlying operating systems, the model is very attractive for implementing distributed applications in heterogeneous systems [6, 17].

Reduces the network usage Using different techniques (e.g. processing and compression before transmission or filtering the data before transmission according to the business logic) the mobile agents can importantly reduce the size of the transmitted data over the network [16, 6, 27, 17].

Enables real-time monitoring Real time monitoring of remote systems over the network often demands expensive network resources. Furthermore, in time-critical applications, monitoring from a remote system could prove not adequate. The mobile agents paradigm enables the monitoring code to dispatch and install in or near the target computer (the machine we need to monitor), inexpensively monitor for new events, and react in real time [16, 6].

Robustness Due to the independence of the mobile agents, the paradigm can be used to create robust distributed network applications that can handle network failures and still perform well [16, 6, 21].

Low requirements The mobile agents paradigm compared to some other distributed computing paradigms has low hardware/software requirements [6, 27]. This makes the paradigm very attractive for applications involving hardware with limited resources e.g. mobile phones and portable data terminals.

However, as with any evolving paradigm in computer science, the scientific community detected several important limitations and problems of the mobile agents model. The most important of these problems follow:

Security issues Allowing foreign processes to use resources of the machine is very dangerous [31, 27, 6]. The foreign processes may be malicious, and attempt to make damage to the machine or use the network resources of the machine to perform a network attack. Furthermore, even in the case that efficient and secured authentication is supported, launching a third party mobile agents platform in the

machine, which requires opening some of the ports of the machine, practically exposes the machine to many more dangers.

The security issue is the most important drawback for the mobile agents paradigm. Most of the modern mobile agents platforms offer several authentication methods to secure the host computer and its resources from malicious and unsigned code. However, there is still much needed work.

Acceptance from the human factor Mainly due to ignorance but also due to the security issues remaining to be solved, the mobile agents paradigm is not widely accepted from the human factor [31].

Requirement of specific platforms In order for a mobile agent to dispatch to a remote machine, the machine must run a suitable platform to accept the mobile agent [31]. None of the operating systems currently available allows the system to host mobile agents, so, a mobile agents platform must be run in the system. This is the case with all the distributed systems today e.g. CORBA, since these platforms never get in the default functionality of any operating system for security reasons.

Reliability A mobile agent moving to a remote machine is not necessarily moving in a friendly and trusted environment [31]. The remote machine may, in purpose or by mistake, try to kill it, alter or read its data, or modify its functionality. The mobile agent also cannot make any assumptions about the resources available in the remote system.

Performance Due to the languages used for implementing mobile agents(mainly java) systems based in mobile agents often have poor performance [27]. Fortunately this does not remain true with the advances in the java language, but several older implementations of mobile agents (with java 1.1) were significantly slow.

Portability of the mobile agents Some older mobile agent platforms were implemented for a limited number of operating systems [31, 27]. Thus, these platforms were rather unpractical for the real life. Furthermore, some other platforms were supporting OS-dependent mobile agents (i.e. implemented in scripting or system-dependent languages). Finally, there is no cross-platform support between the different mobile agents platforms. Mobile agents created in one platform cannot be hosted in another platform, even if they are both constructed in the same language e.g. java.

Communication protocol The mobile agents paradigm favors communication between the mobile agents. There have been some attempts to construct a communication

protocol, an Agent Communication Language, so that agents constructed from different developers or from different applications can collaborate for a common cause. However, this is still research work, and not supported by any mobile agents platform [27].

2.3 Applications of mobile agents

There have been many reports in the past for using mobile agents for constructing distributed applications. The inherent mobile agents characteristics (mainly the ability to move nearer to the data and independence) can empower many distributed applications and offer more robust and easier solutions. The main application areas of the mobile agent paradigm follow [25]:

Distributed computing The inherent independence that characterizes the mobile agents makes the model very attractive for constructing robust distributed applications [13].

Remote computing In the case of a task that must be performed to a remote computer (e.g. because the computer is overloaded or because the task needs to make some measures on a remote machine), then the mobile agents paradigm can be used.

Periodic monitoring In the case we require periodic monitoring of a remote machine, then it is better to send a mobile agent in the remote machine, in order to make monitoring easier and light for the network. Furthermore, this way we can react faster (in real time) to events.

Disconnected and weakly connected operations In the case of a series of operations that must be performed at a remote machine which is not constantly connected to the network, or connected with weak connection, a mobile agent can migrate to the remote machine and perform the operations there, storing any results in the local memory (e.g. in [41]). Upon reconnection, the results (if any) can be sent back to the central server for further processing.

There are many successful applications powered from mobile agents. For example, DITIS [38] uses mobile agents to support mobile users, and UCYMicra [36] uses mobile agents to crawl the web. Furthermore, Samaras et al. in [40, 37] suggest a number of methods based on mobile agents for optimizing distributed database systems.

Chapter 3

Architecture of a complete search engine

3.1 Introduction

In this chapter we will describe the main parts of a search engine. We will also elaborate on the most important problems that the search engines face and describe several approaches on updating the search engines databases (web crawling).

3.2 Search engines

The web is considered from many scientists as one huge distributed database. This is because the distributed nature of the web approaches the nature of the distributed databases. However, unlike the ordinary databases, the web does not offer any indexing and searching functionalities/abilities. This functionality is not at all supported from the initial proposal for the web and the underlying protocols. The searching and indexing tasks for the web are currently handled from specialized web applications, called search engines. Without them, the web, being so unorganized, rather 'chaotic', could not scale. Up to now, the search engines managed to serve their purpose well, and index enormous numbers of web documents.

There are many publicly available search engines in the web today. These search engines can be categorized in two categories: (a) the search engines that maintain their own database with information from the World Wide Web, and (b) the so called meta-search engines (e.g. MetaCrawler [24]), that perform searches by facilitating services

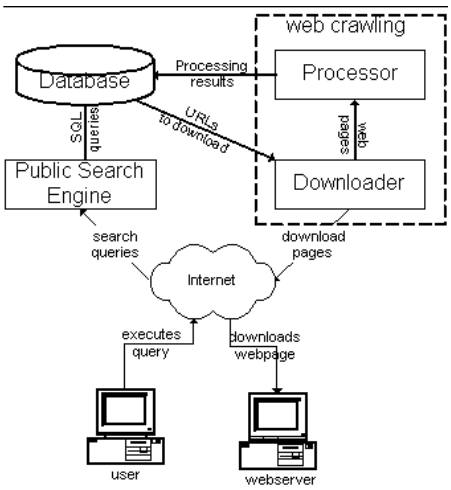

Figure 3.1: A generic search engine architecture

provided from other search engines. More specifically, the former search engines use web crawlers to crawl as many pages as possible (to collect as much data as possible), while the latter answer to the search queries by combining query results from other search engines, with no need to keep their own database with updated information. There are also some combinations of the two approaches in some search engines, used for better results.

Some commercial search engines, in order to increase their users, also enable several value-added services. For instance, Google enables interaction based in web-services. Any programmer can perform queries by invoking a web service from the Google web server. Moreover, most search engines today also index FTP sites and newsgroups, thus increasing in functionality. Furthermore, some of them offer language tools (e.g. searching returns pages only in a specific language) or facilitate an alerting functionality when a page changes.

We will now elaborate more on the architecture of the modern search engines. We will also study some of the main problems faced from the search engines today, and see the various approaches used for updating the data.

3.3 The three parts of a search engine

While most varying in functionality and quality of service, all the modern search engines with databases (not the meta-search engines) can be divided in the three parts visible in figure 3.1:

The publicly available search engine This includes the HTML interface where the users submit their queries, and the mechanism for serving these queries. This part

is very important for the search engines, since this is the only visible part to the end-users. This part is actually more complex than it sounds, but it is completely out of the scope of this work to elaborate on it.

The database The database stores all the crawled data from the web crawlers. The search engine queries the database in order to answer to any user's request. Furthermore, the database feeds the downloader with URLs to download. The information stored in the database are usually updated from the processor.

The web crawling system The web crawling system is the subsystem responsible for maintaining the search engine database and incorporating the changes from the web. Most of the times multiple instances of this component run in parallel, in a farm of cheap computers, connected with very high bandwidth, in order to alleviate the processing bottleneck. The web crawlers download and process as many web pages from the web as possible, by using the standard HTTP protocol. Due to their functionality, the web crawlers often run into severe network bottlenecks.

All the three parts of the search engine communicate with high-speed network connection, in order to minimize delays from the parallel processing. For more flexibility [19, 34], the web crawling system in most of the proposals is architecturally broken into two subsystems, the downloader and the processor, which are described below:

The downloader The downloader downloads the web pages from the Internet and sends them to the processor for further processing. There are many approaches toward optimizing the downloader's functionality, but they are all limited from the bandwidth of the search engine's Internet connection.

The processor The processor receives the downloaded web documents, processes them, and incorporates them in the search engine's database.

We will now look the three subsystems of a typical search engine in more detail.

3.3.1 The publicly available search engine

As already mentioned, this is the only part visible to the end-users. Using a web interface, the users perform their queries and receive their indexed results. The publicly available search engine includes, apart from the web interface, a search mechanism in order to facilitate queries in huge search engine tables, as well as procedures for ranking the results and returning the answers as web pages.

The search is usually performed based on keywords. There are several methods proposed for optimizing the search process, and this is an important aspect of the search

engines, since they receive thousands of queries per second, and they have to search through huge tables before returning the answer [4]. There are also several methods for performing a query-oriented ranking on the web pages, so that the most relevant (with the query) pages are returned first. However, while an exciting research subject, for the purpose of this work we do not need further analysis on the publicly available search engine.

3.3.2 The database

The database is the most important part of our search engine since it contains all the information used for answering the users requests. An updated database would mean better search results. Furthermore, a large database, containing bigger portion of the web's available documents, can give more complete answers compared to a smaller database.

The search engine databases typically have an enormous size (several thousands of terrabytes). Keeping a database of such a size and and enabling fast searching and easy updating from many concurrent processes [20] is not at all trivial. More specifically, any search engine database must offer-enable the following functionalities:

1. Fast searching for the pages that include a (set of) word(s). This is needed for answering the user queries
2. The ability to find all the words that are included in a web page, or a set of web pages. This is needed for the updating procedure
3. The ability to quickly delete, update or insert new records, and at the same time update all the relevant indexes
4. The ability to concurrently serve thousands of users and processes
5. The ability to expand in more than one hard disk drives and machines, due to its enormous size

For this reason the search engine databases are usually distributed in several servers. There are also several tailor-made indexing and searching techniques for optimization reasons.

Usually, instead of using ready-made solutions from third parties (i.e. commercial SQL servers), most of the search engines construct their own database systems according to the required needs [20, 4, 47].

Google Bigfiles: One of these systems (one of the few that are publicly described), is the database used for the Google search engine (a preliminary, academic version). The

database, described in [4] was built from scratch and distributed in several machines. The system enables the fragmentation of the database to several files, that can be hosted in different machines, completely transparent from the user (the user realizes only one database, and only one file. The fragmentation system is called Bigfiles). The files are saved in their own filesystem that optimizes the required functionality, and allows 64-bit addressing (permitting files of up to 17 x 10^9 Gigabytes).

Furthermore, a specific database design enables faster searches. The database contains the `repository`, a table with the complete (compressed) text for every html page crawled, and the `lexicon`, a table that keeps the most important keywords from all the web pages, and assigns them an identification number. Also, the `document` table contains specific information for every stored URL (i.e. the PageRank). The `hitlist` implements the M:M relation between the two tables (`document` and `lexicon`). The `hitlist` table also contains a rank (the importance of the word on each page). Finally, in order to enable faster searching and updating, we have two indexes built in our database, the `forward` index and the `reverse` index. The `forward` index enables fast searching for web pages related to a keyword, while the `reverse` index enables fast updating of the database with data from new or updated web documents.

3.3.3 The web crawler

The web crawling system is the subsystem responsible for maintaining the search engine database and incorporating the changes from the web. The efficiency of the web crawling system is directly analogous to the available Internet bandwidth dedicated to the system.

Multiple instances of the web crawler can run in parallel, in a farm of cheap computers, connected with high bandwidth, in order to alleviate the processing bottleneck. The web crawlers download and process as many web pages from the web as possible, by using the standard HTTP protocol. Furthermore, several optimization techniques can increase the web crawlers' throughput.

Several researchers find an alternative approach more suitable for the web crawling task. More specifically, they structurally separate the web crawler in two distinct components, namely the downloader and the processing component. These components run in parallel, and have high independence between each other. This results to better flexibility and more efficient distribution of the load in the collaborating computers [19, 34]. Furthermore, several other optimizations are more easily adopted in the model, and the throughput is generally improved.

The Downloader The downloader is a very important part of the web crawler. This process uses standard HTTP requests to download web pages from the web. The down-

loader issues a series of parallel HTTP-GET requests to download a set of web pages. The pages are then forwarded to the processor in order to be processed and integrated in the database. While simple in functionality, the downloader is not so simple in implementation. The process of downloading hundreds of pages per second through a single line of communication is not trivial since the performance is limited from the network capabilities. Furthermore, the downloaders have several `good manners` that have to follow, in order to avoid creating any troubles in the targeting web servers.

Some of the optimization techniques that are usually applied to the downloaders for alleviating the network bottleneck follow:

TCP Connection reusability The downloaders keep one TCP connection open for each IP and reuse it until they download the whole contents of the web-server. This results to important time and network savings [4, 19].

Compression Some web servers enable compression(usually with the GZIP algorithm) of web pages before sending them to the requested client. The downloader, as with any HTTP client, may define in its request that it can accept a compressed web page, and perform decompression locally. This significantly reduces the network usage. However, not all the web servers support compression. Furthermore, at most of the cases, compression is disabled from the web server administrators in order to save processing power from the web servers.

Conditional GETs Most of the downloaders perform conditional GET requests. Conditional GETs are defined in the standard HTTP 1.1 protocol [12] (and implemented from most web servers). They force the web servers to return the complete page only if the page is changed after a specific date. In the opposite case, a NOT MODIFIED (HTTP 304 response code) message is returned to the client. This saves important network resources and significant time in the case where the page is not updated since the last crawl [39].

Varying refresh frequency Not all the pages need to be crawled often. Some of the pages (e.g. CNN news network home) get updated every few minutes, while some others do not change for a whole year. Furthermore, some web pages are considered more important than some others (i.e. a university's web page would be more important than a personal web page). For this reason, most of the commercial search engines crawl the important web pages and the web pages that are expected to get updated often (based on previous observations), with higher frequency than the pages updated rarely [32, 46, 10].

DNS caching While the downloading of URLs is the main reason for the network bottleneck, the DNS resolutions also create an important network traffic. Fur-

thermore, a bottleneck in the DNS servers is expected to happen, affecting the throughput importantly [3, 2, 4, 19, 34]. For this reason, advanced implementations of the downloader or the web crawling component include a limited, inexpensive, DNS cache, which however results in significant performance improvement.

Applying the suggested optimizations and several others significantly increases the downloading rate and improves the network bottleneck. However, we have to note that centralized web crawling seems to fail to scale with the web's size, since search engines currently have a very small percentage of the web crawled, ignoring important web pages and missing valuable information. For this reason there were many distributed crawlers proposed up to now that try to alleviate the network and other bottlenecks. Namely, this is why the writers in [4] suggest that the ideal approach for the web crawling problem is the distributed approach. However, previous attempts to a distributed web crawling approach were facing other important issues that prohibited them from being commercially deployed.

The Processing component As already mentioned, the processing component is responsible for extracting the most important information from the web documents and saving them to the database. The most usual approach is to try to extract all the keywords from a web document, and give them an importance rank (based on the position of the word in the page, the page context, and several other heuristics). The occurrence and ranking for these keywords for each page is then saved in the database, to facilitate keyword-based searching.

The processing component performs computationally expensive string processing in order to extract high quality information. Thus, the component often runs into a processing bottleneck, which can be alleviated by providing more computational power in the system, and running multiple instances of it. However, this approach, as with any approach for alleviating any bottleneck in the traditional web crawling system, is prohibitively expensive.

Page Ranking: For any given query of keywords, we expect to find many relevant web pages. Thus, it is important for the users to get the most related and most important web pages first. The relevance of the page with any given keyword is identified during processing of the page (from the processing component), and saved in the database. Moreover, the processing component (or another independent process) defines an importance of each page. For this reason, there are many proposed algorithms, usually based in the number of links showing to each page (backlinks), in the number of pages in each site, and several other heuristics.

The most known algorithm for ranking of web documents is PageRank [33, 4], proposed and used in the original (academic version) Google. PageRank ranks the importance of each page based on the links from other pages to the target page, and the importance of these pages (recursively computed). The PageRank algorithm in Google was executed periodically, and the importance ranks were stored in the database for future reference. The same algorithm was used in many experimental search engines very successfully.

3.4 Current methods for updating the search engine databases

As already mentioned, the most difficult part in search engines is to keep an updated database. The web changes constantly. Pages are added, changed, or removed every day from the web. Cho and Garcia-Molina in [9] detected that nearly 40% of the web change in a period of one week, and in a later study [11] they report an even higher frequency of changes.

The initial proposal in the search engines (abandoned a long time ago) was to request from the web page owners to file their web page with a set of keywords in the search database manually. Each web page developer was manually submitting the web page URL and a set of characterizing keywords to the search engine by using a simple web form. Similar in practice, the web page developer would re-submit the set of keywords in case of a web page being changed importantly. However, soon this kind of update proved impractical and inefficient. Not all the page developers were submitting their URLs manually, and, most importantly, many pages were created dynamically, or changed very often. Furthermore, the number of search engines was increasing and the web developers (or the owners of the pages) could not update all of them.

The trend for keeping the search engines updated is using web crawlers, software that can download and process a mass amount of web pages, and store the processed results (usually in form of keywords) in the database.

3.4.1 Traditional and parallel web crawling

For the last decade, single-sourced (or centralized) Web Crawling has been the driving force behind the most popular commercial search engines. Most search engines perform a local parallelization of the web crawling task by keeping a farm of web crawlers running in local machines. The web crawlers (as described in section 3.3.3) are programs that keep downloading URLs via the Internet, processing them in order to extract the keywords or other important data, and integrating the results in the search engine databases.

The Google [22] and the AltaVista [43] search engines employ an incremental farm of local running crawlers in order to reach a daily dose of a few hundred millions downloads, while the allocation of more computational resources is often announced. However, with the great expansion of the web, especially in the past four years, the traditional crawling model appears inadequate to adjust to the new facts since crawlers are no longer able to download documents with the daily rate required to maintain an updated index of the web. A relatively recent survey suggests that no search engine succeeds coverage of more than 16% of the estimated web size [28]. More specifically, the traditional crawling model fails for the following reasons:

- The task of processing the crawled data introduces a vast processing bottleneck at the search engine site. Distributed processing of data (at the remote Web servers), which would have alleviated the bottleneck, is not available through the current HTTP protocol used by crawlers. Current practices to alleviate this bottleneck are focused in the addition of more computational resources. However, the latter complicates resources coordination and increases the network traffic and cost.

- The attempt to download thousands of documents per second creates a network and a DNS lookup bottleneck. While the latter bottleneck can be partially removed using a local DNS cache, the former can only be alleviated by constantly adding more network resources.

- Documents are usually downloaded by the crawlers uncompressed increasing in this way the network bottleneck. In general, compression is not under full facilitation since it is independent from the crawling task and cannot be forced by the crawlers . In addition, crawlers download the entire contents of a document, including useless information such as scripting code and comments, which are rarely necessary for the document processing.

- The vast size of the Web makes it impossible for crawlers to keep up with document changes. The revisiting frequency for non-sponsored documents, in some of the biggest commercial search engines, varies from 4 to 6 weeks. As a result, search engines deliver old content in search queries.

Moreover, the current crawling model has negative consequences for the complete Internet infrastructure:

- The simultaneous execution of many commercial crawlers generates huge non-user related Internet traffic and tends to overload public Web servers, Domain Name Servers, and the underlying Internet infrastructure (routers and networks).

- Not every crawler employs time realization. At peak time, web-servers may be 'bombed' with HTTP GET requests generated by a Web crawler. Consequences may vary from a simple delay to a complete denial of service.

In parallel crawling [8], the crawling units run within the local environment of a single company (in contrast to distributed crawling described later, where the crawlers are distributed outside the company). Parallel crawling is a natural evolution toward accelerating crawling.

Mercator [19], an experimental parallel crawling architecture that was used later on in AltaVista's Search Engine 3, was attacking the scalability problem of crawling by adding more parallel crawlers on the local network. In addition, parallel crawling techniques have been used in the academic version of Google, while sufficient research has been done for more efficient task scheduling and distribution of the crawling task in [8].

Despite the falling prices in hardware and lower connectivity rates, however, parallel distribution of the crawling task within local resources fails to keep pace with the growing Web due to its centralized nature. Downloading of documents and DNS lookup share the same network resources. In addition, coordination of the crawling units and resource management generate a significant network overhead.

3.4.2 Distributed web crawling

To resolve the problems of parallel crawling, non-local distribution of the crawling units was introduced. The first well-documented research dates in early 1994 with the Harvest Information Discovery and Access System [2, 3]. The Harvest prototype was running gatherers (brokers) at the information sources sites (namely the Web servers). Gatherers not only collected data through HTTP and FTP, but they also filtered and compressed data before transmitting it to a centralized database. Unfortunately, while similar software is still used for efficient in-house crawling, the Harvest project failed to become accepted due to lack of flexibility, and administrative concerns.

Grub [29], a recently launched project under the Open-source license, implements a distributed crawling methodology in a way similar to the SETI project [45]. Distributed crawlers (constructed as screensavers) collect (and possibly process) data from local and remote sources and send information to the search engine for updates. However, the control of the whole architecture is centralized since a central server defines which URLs are to be crawled by which distributed crawlers. More importantly, the screensavers model used to implement the Grub crawlers does not actually favor the web crawling application for several reasons: (a) there are security issues, which cannot be easily solved with the screensaver model, since the screensavers always run with the

permission of the user running them, (b) the model did not favor easy updates, and (c) the model did not favor portability to different operating systems.

J. Hammer and J. Fiedler in [18, 15] initiated the concept of Mobile Crawling by proposing the use of mobile agents as the crawling units. According to Mobile Crawling, mobile agents are dispatched to remote Web severs for local crawling and processing of Web documents. After crawling a specific Web server, they dispatch them-selves either back at the search engine machine, or at the next Web server for further crawling. While the model offers speed and efficiency compared to current crawling techniques, important issues are to be resolved such as (a) a more efficient resource management (which is recognized from the writers as important future work), and (b) a more decentralized control of the architecture. In addition, this methodology requires from the mobile agents to constantly move through the Internet since there is no notion of the 'parking agent'. The absence of a parked/stationed agent at the Web server site precludes an immediate way of promptly monitoring the Web server's documents for changes.

The Google Search Appliance [23], a recently launched commercial package by Google, offers local crawling and processing of a company's Web and document servers by plugging into the local network a Linux computer. This hardware/software solution, however, comes at a great cost and installation overhead. Furthermore, a closed Linux box is an approach that could not serve more that one search engine simultaneously, meaning that a similar package from another search engine would demand another similar box plugged in the local network.

Cho and Molina in [8] also present some guidelines and important issues that arise from the distributed crawling model. They state the importance of parallel and distributed crawling, and suggest how an effective distributed web crawler could be designed. They propose and evaluate a number of suitable models. Finally, they propose some metrics for evaluating such an approach.

UCYMicra [36, 35] is another approach for fully distributed crawling. Powered from mobile agents, the UCYMicra approach managed to outperform standard parallel web crawling by requiring one order of magnitude less time to complete the crawling of the same set of web pages. The approach alleviated the processing and the network bottleneck faced from the traditional and parallel crawling, and it was able to scale with the web. Furthermore, the approach, inheriting the distributed nature from the mobile agents paradigm, solved the most important problems prohibiting all the previous distributed alternatives to be commercially adopted. UCYMicra will be described in more detailed in chapter 4.

Chapter 4

The original UCYMicra

4.1 Introduction

In this chapter we present UCYMicra [36, 35], a crawling system that utilizes concepts similar to those found in Mobile and distributed Crawling introduced in [15, 18, 3]. UCYMicra extends these concepts in order to build a more efficient model for distributed Web crawling, capable of keeping up with Web document updates in real time.

4.2 Architecture of UCYMicra

UCYMicra proposes a complete distributed crawling strategy by utilizing the mobile agents technology. The goals of UCYMicra are (a) to minimize network utilization (b) to keep up with document changes by performing on-site monitoring, (c) to avoid unnecessary overloading of the Web servers by employing time realization, and (d) to be upgradeable at run time. The driving force behind UCYMicra is the utilization of mobile agents that migrate from the search engine to the Web servers, and remain there to crawl, process, and monitor Web documents for updates.

UCYMicra consists of three subsystems, (a) the Coordinator subsystem, (b) the Mobile Agents subsystem, and (c) a public Search Engine that executes user queries on the database maintained by the Coordinator subsystem.

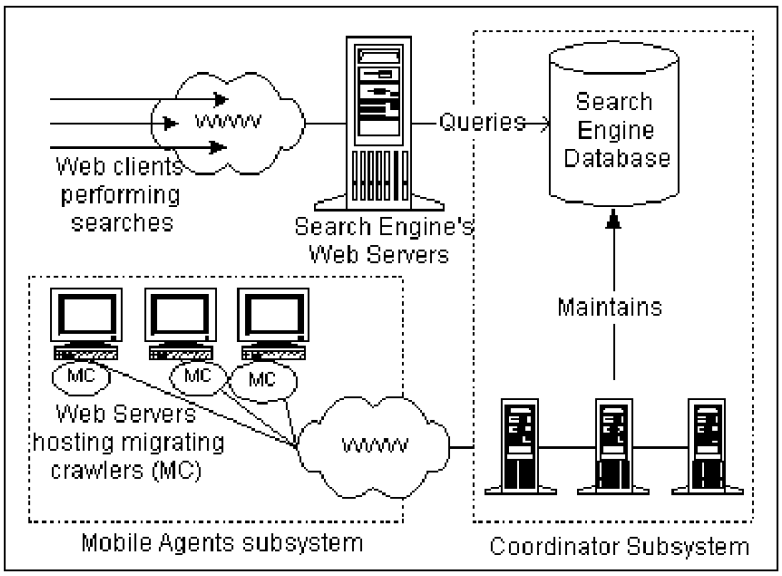

Figure 4.1: Architecture of UCYMicra

The public search engine does not present any research interest, so it will not be described in this work. The Coordinator subsystem resides at the Search Engine site and is responsible of maintaining the search database. In addition, it administers the Mobile Agents subsystem, which is responsible for crawling the Web. The Mobile Agents subsystem is divided into two categories of mobile agents namely the Migrating Crawlers (or Mobile Crawlers) and the Data Carries. The former are responsible for on-site crawling and monitoring of remote Web servers, and the latter for transferring the processed and compressed information from the Migrating Crawlers back to the Coordinator subsystem. Figure 4.1 illustrates the high-level architecture of UCYMicra.

4.2.1 The Coordinator Subsystem

Running locally to the search engine database, the Coordinator subsystem is primarily responsible of keeping it up-to-date by integrating fresh data received from the Mobile Agents subsystem. Second, but not less important, the Coordinator monitors the Mobile Agents subsystem in order to ensure its flawless operation. More specifically, the Coordinator subsystem:

1. Provides a publicly available Web based interface through which Web server administrators can register their Web servers for participating in UCYMicra.

2. Creates and dispatches one Migrating Crawler for a newly registered Web server.

3. Monitors the lifecycle of the Migrating Crawlers in order to ensure their flawless execution.

4. Receives the Data Carriers in order to process their payload and integrate the result in the search engine database.

To avoid a potential processing bottleneck, the Coordinator is implemented to run distributed on several machines that collaborate with a simplified tuplespaces model (similar with Linda's distributed model described in [1]).

4.2.2 The Mobile Agents Subsystem

The Mobile Agents subsystem is the distributed crawling engine of our methodology and it is divided into two categories of Java mobile agents, (a) the Migrating Crawlers, and (b) the Data Carriers (the initial code of both agents resides at the Coordinator System where it is accessible for instantiation and dynamic installation).

In addition to its inherent mobile capabilities, a Migrating Crawler is capable of performing the following tasks at the remote site:

Crawling A Migrating Crawler can perform a complete local crawling either though HTTP or using the file system in order to gather the entire contents of the Web server.

Processing Crawled Web documents are stripped down into keywords, and keywords are ranked to locally create a keyword index of the Web server contents.

Compression Using Java compression libraries, the index of the Web server contents is locally compressed to minimize transmission time between the Migrating Crawler and the Coordinator subsystem.

Monitoring The Migrating Crawler can detect changes or additions in the Web server contents. Detected changes are processed, compressed and transmitted to the Coordinator subsystem.

A Data Carrier is a mobile agent dedicated in transferring processed and compressed data from a Migrating Crawler to the Coordinator subsystem for updating the search database. The choice of using mobile agents for data transmission over other network APIs (such as RMI, CORBA or sockets) is the utilization of their asynchronous nature, flexibility and intelligence in order to ensure the faultless transmission of the data.

4.3 Deployment of UCYMicra

Figure 4.2 illustrates how UCYMicra works. A registration procedure is required for a Web server to participate under the UCYMicra crawling system. The Coordinator subsystem provides the interface through which Web server administrators can define

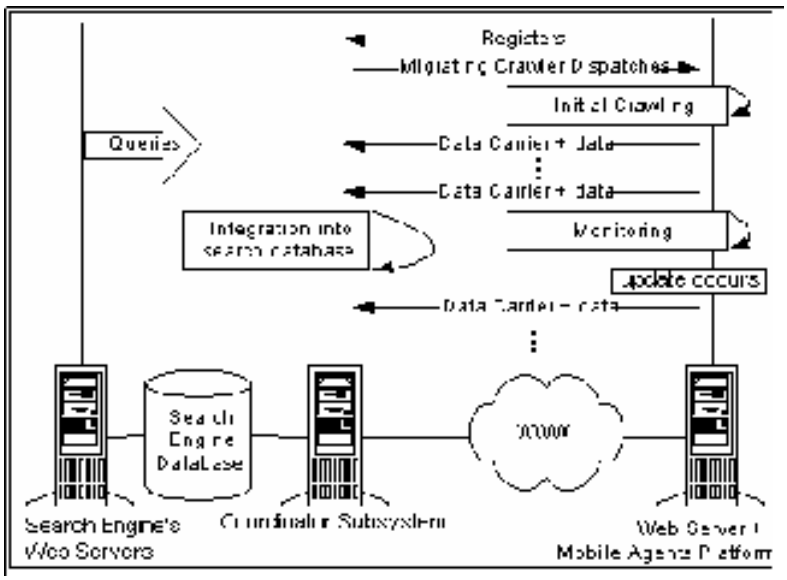

Figure 4.2: UCYMicra lifecycle

the registration parameters. Those parameters are divided into (a) the basic network information of the Web server to be crawled, and (b) the configuration options of the Migrating Crawler and the Data Carriers.

The configuration options provide the Web server administrator with the ability to customize the behavior of the Migrating Crawler prior to its creation and relocation to the Web server site. More specifically, the Web server administrator defines[1]:

1. The time spans in which the Migrating Crawler is allowed to execute. Web server administrators might wish that the crawling and processing of data be executed during off-peak hours.

2. The sleep time between successive crawls used in monitoring the Web server contents for changes or additions. Web servers with frequent changes might consider having a smaller sleep time.

3. The maximum packet size allowed of processed data for the Migrating Crawler to hold before dispatching it to the Coordinator subsystem for integration into the search database.

4. The maximum time-to-hold the processed data (packets) before dispatching it to the Coordinator subsystem. This parameter was introduced in order to avoid the case where processed data may become outdated before dispatching it to the Coordinator subsystem. Data may stall at the Migrating Crawler until the maximum

[1]The Web server administrator can also modify the configuration options at run time, in order to change the behavior of the Migrating Crawlers

packet size allowed is reached, in which case a data carrier is created, assigned the data, and dispatched.

5. Whether the Migrating Crawler will perform the crawling directly on the file system or through HTTP. For the former method, the absolute path(s) of the Web documents is provided. This method is recommended for Web servers with static Web content. For the second method, the Web server's URL(s) is provided and it is recommended for Web servers with dynamic content. Moreover, a combination of the above two can be used.

Following a successful registration of a Web server, the Coordinator subsystem creates a new instance of a Migrating Crawler and updates it with the registration options. The Migrating Crawler is then allowed to dispatch itself to the Web server.

Upon arrival at the Web server, the Migrating Crawler parks and suspends itself until the beginning of a pre-configured time span for execution arrives. For the first crawl, the Migrating Crawler will load, and process all the available Web documents at the Web server. Processing may include removal of useless HTML comments, scripting code, or even local extraction of keywords and ranking for each keyword (based on its occurrence frequency and other properties i.e. position on the document, font size and color). The processed data is being compressed and stored in memory until a data packet with the maximum allowed size can be assembled.

A Data Carrier agent is created for every packet assembled, or when the time-to-hold of a packet expires. Data carriers will then dispatch themselves to the Coordinator subsystem where they deliver their payload.

After the first crawl is completed and the Coordinator subsystem has updated the search database, the parked Migrating Crawler monitors the contents of the Web server for changes or additions of new documents. For documents retrieved through the file system, the last update time is used whereas for documents retrieved through HTTP, a conditional HTTP GET is used (using the If-Modified-Since header [12]). In either case, when the Migrating Crawler detects an update or an addition, it crawls and processes the affected documents, and transmits the new documents' index to the Coordinator subsystem. The transmission follows the rules defined in the pre-mentioned parameters 3 and 4, set by the Web server administrator.

4.4 Security in UCYMicra

Security is always an important aspect in every distributed system. In UCYMicra, security concerns are twofold:

- First, a secure runtime environment must be provided to Web server that will host the Migrating Crawlers. Such a runtime must guarantee that the migrating crawlers have access only to the necessary resources of their host, both software and hardware.

- Second, the runtime must also guarantee that the host machine has no access to the Migrating Crawler's processing code and data, in this way preventing malicious altering of the processing results.

Since the scope of this paper is to provide proof of concept for our crawling methodology, UCYMicra still does not employ security. Mobile Agents Security [48, 49], as well as the protection of the mobile agents themselves [26, 42] is a well-researched topic. It should be straightforward to embed security mechanisms in UCYMicra and this is part of our ongoing work.

4.5 Evaluating UCYMicra

The evaluation of UCYMicra revealed that the new distributed approach was able to outperform the traditional-centralized crawling by requiring significantly less time to crawl a set of web pages. Furthermore, the new approach required to transmit significantly less data (via the Internet) compared to the centralized web crawler.

More specifically, compared to the traditional crawling approach, UCYMicra required one order of magnitude less time to complete the crawling procedure (download, parse, process, integrate to the database). This was mainly for the following reasons:

1. The UCYMicra approach was able to download the web pages in negligible time since the migrating crawlers were hosted in machines connected with high bandwidth to the source machine (the web server). Furthermore, the downloading task was now distributed to the available migrating crawlers.

2. The processing task (processing the web pages to extract the keywords) was done in a distributed nature, thus, alleviating the processing bottleneck faced in the central web crawler.

3. The extra overhead added in the UCYMicra web crawling process, namely, the required time to send the processed data from the migrating crawlers to the central coordinator was significantly less than the time we were saving since we were avoiding the standard downloading procedure. This was mainly because the data was processed and compressed before transmission. Thus, the size of the data was reduced to one order of magnitude, without loosing information.

Furthermore, the new approach required to transmit one order of magnitude less data (via the Internet) compared to the centralized web crawler. This was mainly for two reasons:

1. UCYMicra had the ability to process the web pages before transmitting them and remove things that could not be used from the search engine. Processing removed only information that could not be used from the search engines at any case, like HTML comments, JavaScript code for visual effects etc. Thus, processing did not influence the quality of the search results, and could even enabled us to store a complete text version of the HTML pages in the database.

2. UCYMicra was compressing everything before transmission, thus reducing the data size importantly. The compression and decompression needed negligible time.

4.6 Advantages of UCYMicra

As seen in the previous Section, UCYMicra outperforms traditional crawling without compression by a factor of ten in network utilization and total time required to perform crawling. Besides the performance gains, UCYMicra is a scalable distributed crawling architecture based on the mobile agents technology. More specifically, by delegating the crawling task to the mobile agent units, UCYMicra:

- Eliminates the enormous processing bottleneck from the search engine site. Traditional crawling requires that additional machines be plugged in to increase processing power, a method that comes at a great expense and cannot keep up with the current Web expansion rate and the document update frequency.

- Reduces the use of the Internet infrastructure and subsequently downgrades the network bottleneck by an order of magnitude. By crawling and processing Web documents at their source, we avoid transmission of data such as HTTP header information, useless HTML comments, and JavaScript code. In addition, data is compressed before transmission, which is an option that cannot be forced by traditional crawlers.

- Keeps up with document changes. Migrating Crawlers monitor Web document for changes at their source and updates are immediately dispatched to the Coordinator subsystem. With traditional crawling, several weeks may pass before a Web site is revisited.

- Employs Time Realization. Migrating Crawlers do not operate on their hosts during peak time. As mentioned earlier, our approach requires from the administrator to define the permissible time spans for the crawlers to execute.
- Is upgradeable at real time. Newly developed crawling, processing, or compression code can be deployed over the entire UCYMicra since its crawling architecture is based on Java mobile agents.

Chapter 5

Location aware web crawling

5.1 Introduction

The original UCYMicra proposal [36] did not have the need for location awareness. While the migrating crawlers could easily be enabled to follow links, this was not the immediate target of the methodology. The migrating crawlers were placed in remote sites in order to crawl the servers in the local area network of the host machine, and not a big portion of the web. The crawlers were not allowed to use the network and computational resources of the affiliating company for reasons other than the crawling of the company's data. Therefore, there was no need for efficient delegation of the newly-discovered links. These links, if not belonging to the LAN of the host computer, would not be crawled from the UCYMicra system, but from the traditional crawler working in parallel with UCYMicra.

With the current proposal, realizing the inefficiencies and limitations of the traditional crawling paradigm, we try to make the UCYMicra solution completely independent of the traditional crawling system. Furthermore, we try to delegate the URLs to the available migrating crawlers in an optimal manner so that each URL is delegated to the most near crawler to it for crawling. We call this approach **location aware web crawling** since it is based in the location of each crawler and each web page to make the delegation of web pages to crawlers. Since the UCYMicra system runs over a number of collaborating computers, geographically spread and in different subnets and LANs, it favors the location aware delegation.

Following in this chapter we will elaborate more on the meaning of location awareness for web crawling. Then, we will look into a number of approaches for making a location aware distribution of URLs to our migrating crawlers.

5.2 Location awareness

Web crawling is a well-studied subject in the research community. However, while proposed in the past that each site should be crawled from the most near crawler [8], there is not much work published about the distribution of the web crawling task to geographically spread locations. The current trend in web crawling involves a farm of crawlers working in parallel and using a high-bandwidth connection to the Internet. This way, all the crawlers are located in the same geographical area, and they all use the same network link to connect to the Internet. Subsequently, this introduces very difficult network and processing bottlenecks in the search engine companies, which often requires very expensive solutions.

Realizing these problems, we recently suggested a geographically distributed web crawling method, with the use of migrating crawlers. UCYMicra, powered from the mobile agents paradigm, was performing the web crawling task in a completely distributed manner. The migrating crawlers were deployed to various nodes in the Internet, which belonged to friendly universities and affiliated companies. All the nodes were network-independent of each other (they were not sharing network resources), and geographically distributed.

However, as already noted, the original UCYMicra system was not favoring optimal delegations of the newly-found URLs to the migrating crawlers. While the original system could easily be extended to follow new URLs, the URLs would not be delegated to the migrating crawlers based on their proximity-nearness to each crawler. Instead, a new URL would be delegated to one random crawler. As a result, the crawling would still be done distributed, but not in the optimal manner.

The high distribution of the migrating crawlers in the original UCYMicra system favors a location aware delegation of the URLs to the crawlers. **Location aware web crawling** is distributed web crawling enhanced with a mechanism to facilitate the delegation of the web pages to the distributed crawlers so that each page is crawled from the nearest crawler (i.e. the crawler that would download the page the fastest). **Nearness** and **locality** are always in terms of network distance (latency) and not in terms of physical (geographical) distance. The distinction between the two types of distances (network and geographical distance) is important, since they are not always analogous.

To our knowledge, location awareness for web crawling was not studied in the past, since geographically distributed web crawling was not receiving the appropriate atten-

tion from the scientific community. However, since UCYMicra enables geographical distribution, location aware web crawling appears to be very important for optimization purposes and for reducing the network overhead that occurs during the crawling.

5.2.1 The need for location awareness

Traditional web crawling facilitates standard Internet resources to run and uses the `HTTP/GET` and conditional `HTTP/GET` commands to get the information from the web pages. Since these commands, and their results, are routed through the standard Internet infrastructure, they are routed via the standard routers available in the Internet. Depended on the distance between the source (web-crawler) and the target (web-server) of the `HTTP/GET` request, and the structure of the underlying network (intermediary routers and connections), a simple `HTTP/GET` command can pass through a number of routers until it reaches its final destination. The command (according to `TCP/IP`) will be broken in network packets, and each packet will autonomously be routed to the target address by the intermediary routers. At the target (web-server) the packets will be composed and form the original command. The same occurs with the results that normally follow the `HTTP/GET` command. This process however introduces a certain load in the intermediary routers, which are responsible for sending the packets from one router to another, until the packets reach the destination IP address. To make things worse, web crawlers issue a huge number of such HTTP/GET requests, and thus, introduce an important workload in the intermediary routers, and in the Internet backbone.

Furthermore, as we already demonstrated, the search engines currently have an important network bottleneck in catching up with the web's changing rate. Thus, it is considered important for the search engines to accelerate the downloading process in order to free the network resources as fast as possible.

For all these reasons we suggest to perform a location-aware delegation, where each URL is crawled from the most near crawler. We argue that this convention will result in reduction to the load in the Internet infrastructure caused from the web crawlers. Moreover, it will also alleviate the most restrictive network bottleneck in the crawlers, resulting in higher page throughput in the crawlers.

5.2.2 Evaluation of location aware web crawling

Enhanced UCYMicra In order to evaluate location aware web crawling we should compare it with another distributed web crawling system. Since the original UCYMicra proposal was not sharing the exact same purpose with distributed crawling (with UCYMicra's migrating crawlers we were trying to crawl only the local web documents - in the host LAN), we extended the UCYMicra system so that new links can be followed.

More exactly, in the enhanced UCYMicra (which will be used in the rest of this work for evaluating the proposed approaches), a server administrator hosting a web crawler is not necessarily registering a list of web sites for the migrating crawler to crawl. Instead, the migrating crawler receives the web sites to be crawled, apart from the local ones (if the server administrator registers any), from the coordinator. The coordinator, always respecting the preferences set from the host administrator (time-span, maximum size of crawled data etc) randomly assigns all the URLs from the search engine's database to the migrating crawlers.

The enhanced UCYMicra approach proposed here, unlike the original UCYMicra, is able to follow newly-found URLs. It also inherits most of the advantages of the original approach, since it also works in low-usage hours only, and it manages to importantly alleviate the networking and processing bottlenecks from the search engine's site. However, the new UCYMicra has an extra overhead, compared to the original approach, since the migrating crawlers are forced to download web documents from remote machines (not resided in the LAN of the host computer). Nevertheless, this is the case in any proposed distributed web crawling approach that tries to crawl the whole web. In fact, our own version is even better than most of the others, since it performs local processing and compression before transmission of the data back to the database, and manages to reduce the size of the transmitted data by one order of magnitude (as documented in the original UCYMicra, which shares the same processing and compression algorithms). Thus, enhanced UCYMicra can be used as a good reference of distributed crawling for our evaluation of location aware web crawling, which will follow. We will also use the enhanced UCYMicra approach for the evaluation of the HiMicra and IPMicra algorithms, described later.

Evaluation of location awareness Evaluating location aware web crawling, and comparing it with distributed location unaware web crawling (e.g. the enhanced UCYMicra) was simple. Our evaluation had very encouraging results. Location aware web crawling outperformed its opponent, the enhanced UCYMicra, by requiring **one order of magnitude less time** to download the same set of pages. More details for the evaluation scenario and the experiment setup can be found in section 8.2.

Moreover, from the data collected in this experiment, we were able to reach to some important conclusions. While each web page could be downloaded from any of the migrating crawlers (we used four crawlers), some of the crawlers were able to download the web page much faster than some others. Furthermore, the fast crawlers were not the same for all the web pages. While some web pages, for example, were downloaded faster from crawler 1, some others were downloaded faster from crawler 2. Similarly, others were downloaded faster from one of the other two crawlers. The distribution

of the URLs to the four crawlers was not equal either. The two web crawlers were collecting most of the URLs, the third was collecting a lot less, and the fourth was collecting very few, most of which however were very slow to download from the other three crawlers, and very fast from the fourth. This showed that there may exist logical neighbourhoods in the Internet, which will be most useful if we are able to discover and use for the delegation of the URLs to the crawler.

5.2.3 The need for probing

Location aware web crawling, since the introduction of classless IP addresses, cannot be based solely on the IP address of the two peers, the client and the server. Even if the client and the server have neighbouring IP addresses i.e. 128.122.1.1 and 128.122.1.2, they are not necessarily neighbours in the real world. And even if they are neighbours in the real-life, i.e. they both reside in New York, this does not imply that they are logically near (in terms of network latency) and they can communicate with each other fast.

As a result of this, we need to find the logical distance, which we argue that will help us to build a location aware web crawling system. The **logical distance** between two Internet nodes will represent the latency between the nodes (the time that takes for a packet to arrive from the sender node to the receiver node). We call the task of finding the logical distance between two nodes **probing**.

Probing would ideally be done with the `HTTP/GET` function, since our target is to crawl each site from the crawler that would perform the downloading (`HTTP/GET` function) faster. However, this would demand too much time and have a big network overhead, since the crawlers would have to completely download a web page for probing. In this work we use two alternative metrics for probing, which have results with accuracy very close to the accuracy given from the `HTTP/GET` command, but with significantly lower network and time overhead: (a) the round-trip time between the two nodes, which we can easily measure with the invocation of a `ping` command from the operating system, and, (b) the time required for the execution of an `HTTP/HEAD` request. We found these two metrics to be well-suited for representing the logical distance between two Internet nodes (the migrating crawler that has the lowest logical distance for a site - calculated with `ping` or by timing `HTTP/HEAD` - is expected to be the one that can download that site faster from all the available crawlers). Both these metrics had very similar results when applied to location aware web crawling, and they are very light for the Internet infrastructure. They are also widely supported and accepted from the human factor, and we do not expect their use to create any problems. In order to be OS-independent, we could implement our own code that sends an ICMP ECHO message to the target PC. We could even implement any other suitable lightweight com-

mand and time it to estimate the logical distance, i.e. issue a daytime request (port 13 on TCP/IP). We expect these approaches to have very similar results with the ones we had with `ping` and `HTTP/HEAD`.

During our evaluation, we found the `ping` result from one Internet node to another to be a good estimation of the logical distance between the two Internet nodes. This practically means that, if we download the same site from a number of different nodes in the Internet, we expect the site to be downloaded faster in the node that has the lowest `ping` time. We had the same encouraging results when timing an `HTTP/HEAD` request. In fact, our evaluation (section 8.3) clearly showed that for more than 90% of the pages in the test set, the crawler with the smallest probing time was the same with the crawler with the smallest download time. So, instead of using `HTTP/GET` for estimating the logical distance between two Internet nodes, we can confidently use `ping` or `HTTP/HEAD`, which use less network resources, and introduce less computational workload on the two nodes. The ability to efficiently model, with `ping` and `HTTP/GET`, the logical distance between two Internet nodes, namely the candidate crawlers and the target web page, will empower our effort to cluster the URLs around the crawlers for location aware crawling. We report more on the evaluation of the suitability of `ping` and `HTTP/HEAD` for estimating the logical distance in section 8.3.

Realizing the potentials of location aware web crawling and the suitability of the suggested probing methods for performing a location aware delegation, we developed a number of approaches for making a location aware delegation of the web pages to a distributed crawling system i.e. the enhanced UCYMicra. We will now present these approaches. While we will use UCYMicra for example of a distributed web crawling system where needed, the value of the location aware approach is not limited to the UCYMicra web crawler. Location awareness can be combined with a number of other distributed crawling approaches, which however are currently undocumented (commercial) like Google's Search Appliance and Grub, or still academic work like [18]. In the rest of this chapter we will analyze the brute-force location aware approach, while in the following two chapters we will present two alternative algorithms for location aware web crawling, HiMicra and IPMicra.

5.3 Brute-force location aware web crawling

The concept behind location aware web crawling is simple. Instead of delegating the sites randomly to one of the distributed web crawlers, we now identify the most near web crawler for each site, and assign the site to it. Conforming with this concept, the initial extension in the enhanced UCYMicra system toward location awareness aimed to perform probing of each URL from every existing crawler prior the URL's assignment

to one of the crawlers. Then, the crawler that would appear (from the probing results) most near to the new URL would get the URL for crawling and monitoring for changes (as in the original UCYMicra proposal). This way, each URL would be crawled from the migrating crawler logically (in terms of network latency, in contrast to physical distance) nearer to it.

Straightforward in implementation, this approach demanded every registered migrating crawler to probe any new domain (i.e. by pinging the web-server), and report to a central server the logical distance discovered (the time demanded for the probing). Finally, a central server would find the optimal domain delegation that would not create bottleneck in any of the migrating crawlers and make the most efficient network usage.

This implementation, as expected, had impressive results (presented in section 8.4), and managed to reduce the crawling time importantly. However, it was abandoned soon after the initial evaluation, since the overhead in the network infrastructure for probing all the domains from all the crawlers was high. While in our case this was acceptable since we only use four distributed crawlers and 1000 URLs, in the real Internet environment, with billions of URLs, and, ideally, hundreds of distributed crawlers, performing a brute force delegation would be impossible.

5.3.1 Method optimization

While the method can ensure an optimal delegation, it creates an important number of probing messages (`ICMP pings`, `HTTP/HEAD` requests, or whatever we choose to use for probing). We can avoid the need for probing each URL from all the agents with various optimizations. First of all, since multiple web sites, can be hosted in the same web-server, we can avoid some probings by checking if the same IP address of the web server hosting the URL is already probed in the past. For this reason, we store the probing results in our database. The probing results do not remain perfectly correct during time, but they do not change often either. More specifically, the probing results do not change unless the underlying route that the network packets follow changes drastically, which does not happen very often. Furthermore, we can detect any important change in the probing results, during crawling of the web pages, by timing each download from the web server. In a case we detect such an important change during downloading, we can return the URL back to the coordinator in order to update the database and re-delegate the URL.

Moreover, targeting to less probes, we can ask from the crawlers to probe the new URL sequentially and not in parallel, and stop when we get a probing result less than a threshold. Thus, we will not be able to guarantee the best delegation for all the URLs, but, we will have a sufficient (less than the threshold) delegation, and at some of the

times, with much less probes than the original brute force method.

5.3.2 Conclusions from the brute-force location aware web crawling

To sum up, the advantages of the current model compared to the enhanced UCYMicra model (which was delegating any new links randomly to the available migrating crawlers) are obvious, and very important. We manage to keep the advantages of the UCYMicra system, described in 4. Furthermore, with this extension, we manage to follow newly-discovered URLs in an optimal way, since we crawl each site from the nearest crawler, in the time that this crawler is allowed to work. This results to further alleviation of the network bottleneck in the sites that host our migrating crawlers, and to faster crawling of the delegated sites. Furthermore, since we now enable the crawlers to follow newly-discovered links, we have a stand-alone version of UCYMicra, which does not need to collaborate with a traditional crawler to work (as in the original UCYMicra version). Moreover, the fact that each site is crawled from the (logically) nearest crawler is expected to decrease the load introduced in the Internet backbone due to the crawling task.

However, as already noted, the brute force location aware delegation has an important disadvantage. The process of delegation introduces an extra workload in the Internet infrastructure, due to the subsequent pings that are triggered prior each URL delegation. And even if this workload is significantly less than the complete workload saved from the original UCYMicra solution or any traditional crawling solution, we feel that there could be important improvement to this, and we will shortly propose other ways toward that.

5.4 Conclusions

In this chapter we introduced the meaning of distributed location aware web crawling. Our experiments with location aware web crawling revealed that this approach can importantly reduce the required downloading time, thus, optimize our distributed crawling approach and reduce the time spent in the downloading function by one order of magnitude. We also tested an implementation of the location aware web crawling which was using a probing function to estimate the distance between two Internet nodes (in order to find the nearest web crawler for each URL), and had very promising results.

However, while improving the crawling throughput importantly, the brute force implementation of distributed location aware web crawling required an extensive number of probes, resulting to a significant network overhead. Thus, we now need to find other location aware web crawling algorithms that have less network overhead.

Chapter 6

HiMicra

6.1 Introduction

At the previous chapter we presented the importance of location aware web crawling. We also presented a simple system for location aware web crawling, based on brute force location aware delegation. Brute force location aware web crawling was able to make an optimal delegation of the web sites to the migrating crawlers, resulting to larger downloading throughput, and eventually, best search engine results. However, our method was generating an important number of probes for each new IP address. For this reason, we now propose an alternative method that has satisfactory results without the need of exhaustive probing. The new method is based on a hierarchy of migrating crawlers, and it is called HiMicra. The method manages to reduce the number of probes needed to an order of magnitude. Paying the cost of less network overhead, we cannot guarantee the most optimal delegation of the domains (as we could easily do in the initial extension), yet, we can easily witness efficient URL delegation.

6.2 Overview of HiMicra

The previous algorithm for location aware delegation results to the optimal delegation of the URLs to the migrating crawlers. Running that algorithm, we make sure that each URL is going to be crawled from the migrating crawler that is *logically* most near to it. However, the execution of the brute-force method requires extensive pinging. More specifically, an unoptimized implementation of the algorithm would require all the migrating crawlers to probe (by pinging or by measuring the time for an HTTP/HEAD request) each URL prior the URLs delegation, this resulting to a sufficient network overhead. This lessens the value of the brute-force location aware delegation as described in the previous section.

At this extension of the UCYMicra system, called Hierarchy based Micra (HiMicra for short) we are specifically targeting in the facilitation of an efficient delegation of newly-discovered domains at the migrating crawlers. We have designed and built an efficient algorithm to create and maintain a hierarchy of migrating crawlers, so that domain delegation could be done fast and with low network overhead. Paying the cost of less network overhead, we cannot guarantee the most optimal delegation of the domains (as we could easily do in the initial extension), yet, we can easily witness efficient URL delegation.

Hierarchy in HiMicra is built in a network-oriented approach (that is, using parameters taken from the network). Migrating crawlers are organized in branches, in network neighborhoods, so that each group of siblings is expected to have similar network latency in connecting at the majority of the available network places in the Internet. The similarity factor in siblings is weak at the first level, and growing as we move to lower levels in the hierarchy. The parental relationship declares that the parent migrating crawler could be a sibling with the children migrating crawlers (in terms of network latency), but is taking the parental place just for matters of organization (so that any communication from a higher level is directed only at him, and not at all the group).

The innovative in the described hierarchy is the building and maintaining mechanism. We developed an algorithm that enables an easy and highly efficient grouping of the migrating crawlers with low overhead. As expected, our algorithm cannot guarantee the most accurate hierarchy, which will result in the optimal delegation, but it can build a very efficient structure with a non-prohibiting way (negligible overhead).

The coordinator uses this hierarchy in order to probe any newly discovered domain (if the domain or the domain's IP address was never delegated in the past). In fact, we use a best-first search algorithm, in order to follow the most promising path, which we expect to lead us to a crawler with an acceptable probe. The algorithm is described in detail in a following section.

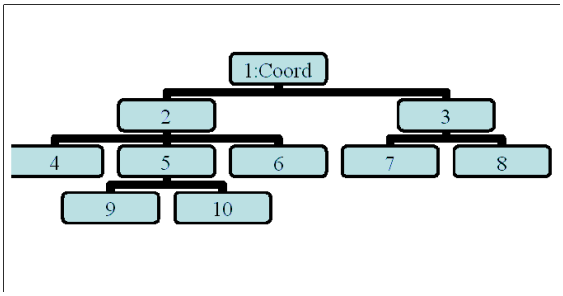

Figure 6.1: Simple HiMicra hierarchy example. Hierarchy in HiMicra is built in a network-oriented approach. In this example, we are expecting that migrating crawlers 4, 5 and 6 will have a low variation in their network latency in communicating with most of the internet hosts. Migrating Crawlers 9 and 10 are expected to have even less variation in their network latency in communicating with most of the hosts. Coord is the coordination subsystem, which only forwards any messages, and acts as a parent for migrating crawlers 2 and 3.

The UCYMicra extension continues to have the same three subsystems that the original UCYMicra system had (a) the Coordinator subsystem, (b) the Mobile Agents subsystem, and (c) a public Search Engine. However, only the search engine subsystem remains without a change.

6.3 The Coordinator system

A small change at the original Coordinator subsystem enables it to find unregistered domains and propagate them at the Mobile Agent subsystem for registration at one of the migrating crawlers. Moreover, the task of monitoring all the migrating crawlers for any malfunctions is no longer a responsibility of the Coordinator subsystem. Instead, the hierarchical organization implemented in the Mobile Agents subsystem facilitates the of all the migrating crawlers in a simpler way.

More specifically, the Coordinator subsystem:

1. Provides a publicly available Web-based interface through which Web server administrators can register their Web servers (that will host the migrating crawlers) under the UCYMicra.

2. Uses probing data from the migrating crawlers in order to construct a hierarchy of them. The hierarchy is built in a batch mode (i.e. once a day) and the migrating crawlers are then notified for their place in the hierarchy.

3. Receives and processes the Data Carriers. The Coordinator subsystem enables the Data Carriers to park and suspend locally until the processing time of their valuable data has arrived.

4. Monitors the lifecycle of the migrating crawlers in the Mobile Agent subsystem hierarchy, creates new migrating crawlers or terminates existing.

5. Locates domains that are not registered at any migrating crawler and forwards them for further delegation at the mobile agent subsystem. Also, it keeps track of the domain delegation which is done in the Mobile Agent subsystem, so that it can identify which domains are registered at each migrating crawler, and which are not registered anywhere.

The coordinator module is now responsible for discovering the new domains and coordinating their delegation to the available migrating crawlers. In order to avoid any bottleneck, we use collaborative components ran in more than one machines in parallel to implement the module. For communication of these instances we use tuplespace-like techniques as described in [1].

6.4 The Mobile Agents subsystem

The Mobile Agents subsystem is the distributed crawling engine of our methodology and it is divided into two categories of mobile agents, (a) the Data Carriers, and (b) the Migrating Crawlers (the initial code of both agents resides at the Coordinator System where it is accessible for instantiation).

A Data Carrier is a mobile agent dedicated in transferring processed and compressed data from a migrating crawler to the Coordinator subsystem for updating the search database. The choice of using a mobile agent for data transfer over other network APIs (such as RMI, CORBA or sockets) was based on the flexibility and autonomy offered from the model compared to the other alternatives.

A Migrating Crawler on the other hand is the one responsible for dispatching to the remote host and crawling the delegated sites. In addition to its inherent mobile capabilities [7], a Migrating Crawler of the original UCYMicra model was capable of performing the following tasks at the remote site:

Crawling Given a starting page or URL, a Migrating Crawler can perform a complete local crawling either though HTTP or using the file system.

Processing Downloaded Web documents are stripped down into keywords, and keywords are ranked based on widely used algorithms.

Compression Using Java compression utilities, the extracted keywords lists are compressed to minimize transmission time.

With the HiMicra extension, the Mobile Agents subsystem is organized into a hierarchy, enabling easy and fast delegation of the domains at the mobile crawlers. The

hierarchy is built in a way where neighbouring migrating crawlers are expected to have very similar network latency with each domain. Subsequently, migrating crawlers in the extended UCYMicra model are also able to:

Probe a domain Probing of a domain is done to calculate a logical distance (network latency) from the web-server hosting the domain to the migrating crawler's location. This may be easily succeeded through pinging.

Time the downloading task The migrating crawlers can time each download, in order to discover if there are important changes in the Internet infrastructure that should trigger the re-delegation of an already delegated site. Such changes, while rarely happening, are detected from the migrating crawlers, which then decide if they should ask for re-delegation for the affected sites.

6.5 Constructing the hierarchy

As already demonstrated, our initial attempt to probe a site from all the migrating crawlers in order to find the most suitable crawler (most near in network distance) was rather expensive, since the propagation demanded probing from all the migrating crawlers. The current version is able to drastically limit the demanded probes importantly by facilitating a hierarchy of the available migrating crawlers. Furthermore, the construction of the hierarchy is not expensive either, since we only need to probe (from all the crawlers) a limited sample of the Internet sites (i.e. 10000 IP addresses), and not all the discovered Internet sites.

The concept of the hierarchy is to try and cluster the crawlers in groups having similar distance from a number of remote sites. URL delegation is then performed with a best-first search algorithm (described in detail later). In this approach we are not after the optimal delegations. The purpose of HiMicra algorithm is to perform near-to-optimal delegations, and, most importantly, avoid the worst case scenarios. Nevertheless, our experimental results show that with a reasonable setup we manage to find the optimal solutions in more than half of the domain names, and with less than half of the probes that the brute-force approach would require.

Prior to the construction of the hierarchy, the coordinator module asks from all the crawlers to probe a number of randomly selected (from the crawling database) IP addresses. The crawlers probe all the IPs and report the results back to the coordinator, who is then responsible for constructing the hierarchy. Then, by facilitating these results, the coordinator constructs the hierarchy, so that the crawlers that are expected to have similar probing results for a site, are neighbors in the hierarchy.

The hierarchy is built from top to down, always having as a root the coordinator. The

hierarchy groups the crawlers so that neighboring crawlers in the hierarchy have similar probing results for most of the sites. The similarity factor is weak at the first levels but it grows as we move to lower levels in the hierarchy. More precisely, the hierarchy is built as follows:

1. The coordinator interactively gets the branching factor of the hierarchy from the user[1]. It also initializes the set of available crawlers S_c to include all the available crawlers, and the set of the testing URLs S_t to include a subset of URLs randomly selected from the database (which are going to be used to build the hierarchy). Also, the coordinator sets as C_{root} itself.

2. The coordinator asks all the crawlers in set S_c to probe all the URLs in set S_t and collects the results.

3. For every set of crawlers $S_{avail} = C_1, C_2, \ldots, C_n$, where n is the branching factor and $C_1, C_2, \ldots, C_n \in S_c$, we calculate $DiffFunction = \sum P_{diff}(i,j) \forall$ pair of crawlers $i, j \in S_{avail} \cup C_{root}$,
   where $P_{diff}(i,j) = \begin{cases} 0 & \text{, if } i \text{ or } j \text{ is the coordinator} \\ \sum_{\forall s: site \in S_t} |(P(i,s) - P(j,s))| & \text{, otherwise} \end{cases}$
   where $P(crawler\ \alpha, site\ s)$ is the probing time of site s from crawler α

4. The subset which maximizes the $DiffFunction$ (calculated from the previous step) is chosen over all the possible subsets, and the crawlers in the set are delegated to C_{root} as children (in no particular order).

5. We cluster each of the undelegated crawlers around the new children. We assign each crawler to the cluster of the child that minimizes the distance difference $ddiff(C_{uc}, C_{child})$, where $ddiff = \sum_{\forall s: site \in S_t} |P(C_{uc}, s) - P(C_{child}, s)|$, where C_{uc} is the undelegated crawler and C_{child} is the newly delegated child crawler. The meaning of clustering around a central (already delegated) crawler is that each cluster groups crawlers of very similar distance from all the sites in S_t. Thus, these crawlers will be placed as descendants of the central crawler (the newly-delegated child).

6. For each of the new children, we recursively repeat the algorithm from step 3, this time setting C_{root} as the child and the available crawlers S_{avail} as the crawlers clustered around the child. We stop when all the crawlers are delegated.

[1] In our experiments (due to the limited number of available crawlers), the branching factor was always set to two

We clarify these rules with a simple example. If we have only one possible root crawler (crawler 1) and four possible children (crawlers 2, 3, 4, and 5), and a maximum of 2 children for each parent, then we get the following crawler pairs we have to check for: {2,3},{2,4},{2,5},{3,4},{3,5},{4,5}. Then, we add the root crawler (crawler 1 in our case) in each of these sets, which produces the sets: {1,2,3},{1,2,4},{1,2,5},{1,3,4},{1,3,5},{1,4,5}. We then calculate the $DiffFunction$ for all the sets, and select the set which maximizes the function's value. Suppose that this is the set (1,4,5). We delegate crawlers 4 and 5 as children of crawler 1 in the hierarchy. Then, we cluster the remaining crawlers (crawler 2 and crawler 3) around crawler 4 and crawler 5, so that each undelegated crawler is placed in the cluster of the child (4 or 5) that minimizes the value of $ddiff$ function. Now suppose that both crawlers are clustered around crawler 4 because $ddiff(2,4) < ddiff(2,5)$ and $ddiff(3,4) < ddiff(3,5)$. This means that crawlers 2 and 3 will be descendants of crawler 4. Since we only have two descendants of crawler 4, we immediately delegate them as children of crawler 4 and finish the hierarchy. If we had more than 2 descendants of crawler 4, we would proceed recursively in a similar manner, this time setting crawler 4 as the root crawler and the crawlers clustered around crawler 4 as the available crawlers.

Since the Internet infrastructure, and the underlying connection links are not stable, we need to rebuild the hierarchy often, in order to remain updated with the changing underlying connectivity. Rebuilding the hierarchy from scratch once a month is significant, since the changes in the Internet's underlying structure are not expected to happen very often (they are mostly based in very expensive agreements that do not change very often). Rebuilding the hierarchy does not mean re-delegating all the URLs from scratch. Any URL that was delegated in the old hierarchy will be re-delegated to the same crawler in the new hierarchy (if that crawler is still available) without any probing. Only the new URLs (that were not delegated in the old hierarchy) will need to be probed and delegated according to the HiMicra algorithm.

Building the algorithm to run in a centralized environment (in the coordinator module) and create the hierarchy is trivial and has no research interest, thus not presented here. For our evaluation tests, we built and ran an unoptimized brute-force (makes all the possible placements in the hierarchy and selects the best) algorithm. The algorithm was able to create the required hierarchy that satisfied the pre-mentioned three rules for 50 migrating crawlers and 2000 test sites in an average PC with limited memory (Pentium 4 1.7 Ghz, 256Mb RAM) in less than an hour. An optimized version of the algorithm could include clustering techniques borrowed from genetic algorithms or linear programming, and significantly reduce the execution time for problems of similar size. However, optimization of this algorithm is out of our research interests, since current advances in hardware and other computer science areas (as genetic algorithms and

linear programming) can improve the execution time.

6.6 Using the hierarchy to delegate the URLs

HiMicra uses the generated hierarchy to efficiently delegate the new domains to the available crawlers. More precisely, it performs a best-first search in the hierarchy, in order to inexpensively (without many probes) locate a suitable crawler to handle the new domain. A suitable crawler would be a crawler that satisfies a threshold, called **probing threshold**. Probing threshold is the maximum acceptable probing time from a crawler to a page and it is set by the search engine's administrator depending on the required system accuracy. In simple terms, the higher the threshold, the less optimal delegations we get. During our experiments we found a probing threshold set to 50msec to give a good ratio of quality for number of probes.

The coordinator is responsible for executing the best-first search algorithm and requesting the probing of new domain names from the web crawlers. Thus, when the coordinator discovers a new domain name, it first asks from the top-level crawlers (its immediate children) in the hierarchy to probe the new domain. The probing results (with the identity of each crawler) are then added in an ordered list, where smallest probing results occur first. Then, the coordinator removes the first probing result from the list (which is also the smallest). If the probing result is smaller than or equal to the probing threshold, the domain is delegated to that crawler. Otherwise, the coordinator asks from the children of that crawler to probe the domain and add the probing results in the ordered list. The coordinator then continues in a similar manner, removing the first probing result from the ordered list, and probing the children of that crawler, until a crawler with a probing result less than the threshold is found. The search also ends when all the crawlers are probed, and no one is found suitable for the domain name, in which case, the domain name is delegated to the best of the crawlers (according to the probing results).

As we already noted, our approach respects the resources of the systems that host the migrating crawlers. Thus, probing of the URLs from the migrating crawlers cannot be done at real time on-demand. Since the migrating crawlers are only allowed to work in their own predefined time-span (e.g. from 01:00AM to 06:00AM local time), delegating a domain name can take some time (depended on the depth of the hierarchy and the number of crawlers), until the different time-spans for all the crawlers that are to probe the URL occur. However, the time needed to optimally delegate a new URL is not important, as long as our delegation procedure does not create additional load in the Internet or the collaborating machines.

Pseudocode for the delegation procedure for a URL follows:

```
final int VERY_BIG_NUMBER = 10000;
int besttime = VERY_BIG_NUMBER;
crawler bestcrawler = coordinator;
URL u;
OrderedList ol = new OrderedList();
ol.add(<<crawler=coordinator,probingtime=VERY_BIG_NUMBER>>);
boolean sufficientnotfound = false;
while (sufficientnotfound && ol.notEmpty()) {
  ol.orderAscending(probingtime); // order asc on probing time
  <<crawler current, probingtime time>> = ol.removeFirst();

  if (time<besttime) {     // used to find the faster if none
    bestcrawler = current; // of the probes is smaller than
    besttime = time;       // the probing threshold
  }

  if (time<ProbingThreshold) { // i found a sufficient one
    current.delegateURL(u);    // delegate to it
    sufficientnotfound = false; // and stop probing
    break;
  }

  for(crawler c in current.sons()) {
    // add them all to the ordered list
    ol.add(<<crawler=c, probingtime=c.probe(u)>>);
  }
}
if (sufficientnotfound) // then delegate to the fastest
  bestcrawler.delegateURL(u);
```

In order to clarify the algorithm, we present a trivial example. Suppose that we discover a domain name which is not yet delegated to any crawler and that we have a hierarchy similar to the one depicted in figure 6.6. First, the coordinator makes a domain name resolution to find the IP address of the new domain name, and verifies that neither the IP address nor the domain name was earlier delegated to any of the available crawlers. Suppose that the IP address is also not delegated yet. Therefore, the coordinator asks the two top-level crawlers (crawler 1 and 2) to probe the new IP address and return the probing results. When the coordinator receives the two probing results (85 and 80 respectively), it adds them (with the identity of each crawler) to a

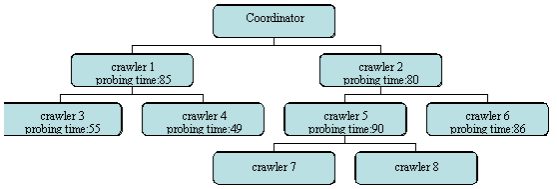

Figure 6.2: HiMicra example

list ordered ascending on the probing results (smallest probing results occur first). The list will now contain the following pairs of $<< probingresult, crawlerid >>$ in this particular order:[$<< 80, 2 >>, << 85, 1 >>$]. Then, the coordinator removes the first probing result from the list, in order to use the crawler which that result belongs to, for further probing (this would be crawler 2 in our example). The two children of that crawler are now asked to probe the new URL and return the results back to the coordinator, which again adds them in the ordered list. The list is now:[$<< 85, 1 >>$ $, << 86, 6 >>, << 90, 5 >>$]. The coordinator then, since all the probing results discovered so far, are larger than the probing threshold, repeats the probing procedure. Again, it removes the first probing result from the ordered list, which would be the probing result for crawler 1, and probes the two children of that crawler. As soon as the probing results from crawlers 3 and 4 (55 and 49 respectively) are received back to the coordinator, they are again added in the ordered list. The list now contains: [$<< 49, 4 >>, << 55, 3 >>, << 86, 6 >>, << 90, 5 >>$]. Then, the coordinator removes the first of the probing results from the ordered list (that would be 49, for crawler 4), which is smaller than the probing threshold. Thus, the probing procedure stops, and the new domain name is delegated to crawler 4.

While in the previous example we had to probe the domain name from nearly all the crawlers (only crawler 8 and crawler 9 did not probe the domain name), in a real Internet hierarchy we can save significantly more probes. Our evaluation results (presented later) show that with 12 crawlers in the real Internet infrastructure we were able to save as much as half of the probes, while with 50 crawlers (generated data) we were able to avoid more than the 4/5 of the probes.

6.7 Dynamic calibration of URL delegations and load balancing

Internet is one of the most dynamic infrastructures in the world. Nodes are added or removed all the time, communication links are inserted or broken, Internet's underlying networking architecture constantly changes. Thus, any approach operating in a network-

oriented manner over the Internet should have a provision for these changes.

While not implemented in our approach, HiMicra can enable dynamic calibration of URLs in order to keep the URL delegations always near-to-optimal. In fact, the HiMicra hierarchy favors dynamic calibration of the URL delegations, since the hierarchy is built in a network oriented manner. More specifically, if, for any reason, a crawler stops having strong connectivity with a URL i.e. downloading of the URL delays more than it used to, then we can try to reassign the problematic URL to one of the neighboring crawlers (children or sibling). Since we expect the neighboring crawlers in the hierarchy to have similar probing results for most of the URLs, there is a great chance that the neighboring crawler is still near to the URL. In the case where neither the neighboring crawler manages to probe the URL significantly fast, then the algorithm can try to re-delegate the problematic URL normally, with the best-first search algorithm already demonstrated.

Similarly, we can enhance HiMicra to perform load balancing, in order to avoid any possible bottleneck in the migrating crawlers' sites. If a crawler is overloaded with URLs, then, the coordinator can remove some URLs from it, and reassign them to the neighboring crawlers (children and siblings), expecting that they will also be able to crawl the URLs significantly fast. Again, if the coordinator fails to find any crawler that can take the URL (is not overloaded and has a probing result lower than the probing threshold) then the URL can be delegated following the original HiMicra algorithm.

While both these enhancements will have significant impact in the algorithm, enabling it to become commercial, we did not evaluate them. The evaluation of these enhancements would demand a significantly bigger data set than the one we managed to collect (12 crawlers and 1000 URLs).

6.8 Evaluation of HiMicra

HiMicra, as a distributed crawling approach, has important advantages compared to a centralized crawling approach. Moreover, compared to other distributed crawling approaches (e.g. the enhanced UCYMicra presented in 5.2.2), HiMicra importantly reduces the required time to download the sites, since each site is expected to be crawled from the most near (or at least a significantly near) crawler.

HiMicra and centralized crawling: HiMicra, as any other distributed crawling approach, has important advantages compared to a centralized crawler. HiMicra easily outperforms any centralized crawler since it distributes not only the downloading task, but also the processing task, to a number of remote machines. While having an extra network overhead of having to transmit the processed and compressed data from the migrating crawlers back to the coordinator, at the bottom line, this overhead is negligi-

ble compared to the benefits of the distributed approach. Since the data is transmitted back to the coordinator compressed and processed (1/10 of its original size [36, 35]), it requires importantly less bandwidth in the search engine site than any centralized approach to crawl (in fact, in our experiments with the original UCYMicra, we did not detect any networking bottleneck in the search engine's site). Moreover, the probing overhead on HiMicra is also negligible, and does not create any bottlenecks, since probing is distributed in the migrating crawlers. Finally, the task of monitoring for changes is also distributed in the migrating crawlers, and without creating any load in the search engine site.

HiMicra and distributed crawling: HiMicra, compared to earlier approaches of distributed crawling, is better, since it performs a location aware distribution of the URLs to the distributed web crawlers. This location awareness results to an important decrease in the time spent for the downloading task. On the other hand, HiMicra has an extra overhead (compared to earlier distributed crawling approaches) since it needs to perform probing of each site from some of the migrating crawlers before delegating the site to one of them. However, this probing is significantly reduced from the probing demanded in the brute-force location aware distributed crawler (presented in section 5.3). In fact, the overhead in HiMicra from the probing function is significantly less than the benefits from the location aware web crawling.

HiMicra importantly reduces the number of the required probes for making near to optimal delegations. The hierarchical approach facilitated from the HiMicra algorithm enables us to quickly locate the subset of the migrating crawlers that are near to each domain name. More specifically, experimenting with real Internet data (12 crawlers and 1000 domain names), we managed to reduce the number of required probes to 1/2 of the total number of probes required from the brute force approach. Experimenting with generated data (50 crawlers and 2000 domain names), we managed to reduce the number of required probes to 1/5 of the total number of probes required from the brute force approach. Detailed results of these experiments are presented in 8.5.

The slight performance of the algorithm with the real data, compared to the increased performance of the algorithm with the generated data was expected. For practical reasons, we were not able to include more than 12 crawlers in our experiments in the real Internet. Therefore, the resulted hierarchy when experimenting with real data was not favoring the HiMicra methodology (not enough crawlers). On the other hand, experimenting with 50 crawlers (the generated data) would allow the HiMicra algorithm to efficiently facilitate the hierarchy for making near-to-optimal delegations, and save us an important number of probes.

For practical reasons we were not able to actually execute the suggested delegation and compare it with the other possible delegations. Running such a function in the

twelve foreign hosts could be characterized as network attack, and would also take significant time and use important resources, that the collaborating networks could not spare. However, based in the previous experiments (section 8.3 and section 8.2), during which we showed direct correlation of probing and downloading time, we argue that the downloading time for the HiMicra delegation would be very near to the optimal downloading time. Thus, we expect the downloading time to be almost one order of magnitude less compared to the average downloading time with a random assignment of URLs to crawlers.

6.9 Conclusions from the HiMicra system

In this chapter we proposed and evaluated HiMicra, the first practical approach toward location aware web crawling. The HiMicra algorithm builds a hierarchy of migrating crawlers, which enables the coordinator to efficiently locate the subset of the migrating crawlers logically near to a new domain name. The new approach reduces the required number of probes for each URL significantly (compared to a brute-force approach).

Construction of the hierarchy and the actual implementation of HiMicra is trivial. However, for practical reasons, we were not able to test HiMicra with real data. Nevertheless, our tests with generated data revealed that HiMicra can scale in pace with the real Internet. In fact, HiMicra seems to significantly improve its results while more crawlers are added.

The experiments conducted with HiMicra reveal that the approach can significantly reduce the required probes for performing delegations of URLs. However, from the experimental results, we expect that HiMicra can perform significantly better when it runs it the real Internet, with a larger number of available migrating crawlers. Thus, for future work, we plan to run the HiMicra experiment with more migrating crawlers. Also, if the experiment results are still promising, we plan to implement a load balancing and a dynamic calibration scheme that will make our approach more dynamic.

Chapter 7

IPMicra

7.1 Introduction

In the two previous chapters, after realizing the advantages of location aware web crawling, we designed and tested two different extensions in UCYMicra that enable location aware web crawling. The first extension required an excessive number of probes (pings or HTTP/HEAD), while at the second extension we managed to reduce these probes importantly without having significant difference in the quality of the results. Now we propose another extension for location aware web crawling, which reduces the need for probing much more, and gives better delegations. The new proposal facilitates the use of information provided from the four RIRs (Regional Internet Registries) to create a hierarchy of IP subnets. This hierarchy is later used for location aware web crawling with even less need for probing. The new method reduces the need for probing by a factor of four.

7.2 IP based hierarchy for location awareness

With IPMicra, we specifically target in the reduction of the required probes for a delegation of a URL to the nearest crawler. We designed and built an efficient self-maintaining algorithm for domain delegation with minimal network overhead by using information from the Regional Internet Registries (RIRs for short).

The basic idea behind IPMicra is the organizing of the IP addresses, and subsequently the URLs, in a hierarchical approach. More specifically, we use the `WHOIS` data collected from the RIRs to build and maintain a hierarchy with all the IP ranges (IP subnets) currently assigned to organizations. We then place the migrating crawlers (that were kindly hosted by collaborating organizations) in the subnet hierarchy, according to their IP addresses. Then, we use the hierarchy in order to efficiently find the most near crawler for every new URL, without the need for excessive probes. More to the point, we first place the URL in the appropriate subnet (based on the URL's IP) and then navigate the hierarchy in a bottom-up fashion, probing the crawlers in the order found. We stop as soon as we find a crawler that satisfies a threshold, called **probing threshold**. Probing threshold is the maximum acceptable probing time from a crawler to a page and it is set by the search engine's administrator depending on the required system accuracy. In simple terms, the higher the threshold, the less optimal delegations we get. During our experiments (see sectionr̃efevalIPMicra) we found a probing threshold set to 50msec to give a good ratio of quality for number of probes.

Our evaluation (presented in detail in section 8.6) revealed that the IPMicra system not only outperforms traditional crawling, but, most importantly, significantly improves the performance of distributed (location unaware) web crawling, without adding any significant overhead. In fact, the added overhead due to the probing function is negligible.

Before describing the IPMicra approach in more detail, it is important to give a small introduction for the Regional Internet Registries (RIRs), since we will use data from them for building the hierarchy.

7.3 Regional Internet Registries

Regional Internet Registries are non-profit organizations that are delegated the task of handling IP addresses to the clients. Currently, there are four regional Internet Registries active in the world:

1. APNIC (Asia Pacific Network Information Centre): Covers Asia/Pacific Region
2. ARIN (American Registry for Internet Numbers): Handles the IP addresses for North America and Sub-Sahara Afrika

3. LACNIC (Regional Latin-American and Caribbean IP Address Registry) - Covers Latin America and some Caribbean Islands

4. RIPE NCC (Rseaux IP Europens) - Covers Europe, the Middle East, Central Asia, and African countries located north of the equator

Despite the anarchy currently experienced in the Internet, the four Regional Internet Registries manage to keep an updated database with IP addresses registered in their area. All the sub-networks (i.e. the companies' and the universities' sub-networks) are registered in their regional registries (through their Local Internet Registries) with their IP address ranges. Later on, if the subnet administrator wants to register another block of addresses, again the addresses are expected to be registered under the same organization name in the same RIR.

With the RIRs functionality, a hierarchy of IP ranges has been created in the Internet. We can consider the IP range hierarchy starting from the complete range of IP addresses (from 0.0.0.0 to 255.255.255.255). Then, the IP addresses are delegated to RIRs in large address blocks, and finally, they are sub-divided to LIRs (Local Internet Registries), where then are finally sub-divided to their customers-end users.

7.4 An outline of the IPMicra system

The IPMicra system is also architecturally divided in the same three subsystems that were introduced in the original UCYMicra: (a) the public search engine, (b) the coordinator subsystem, and (c) the mobile agents subsystem (figure 7.1). However, only the public search engine remains unchanged. The coordinator subsystem is now enhanced with the functionalities of building the IP hierarchy tree and coordinating the delegation of the subnets to the migrating crawlers. Moreover, the migrating crawlers are enhanced with code for probing the sites and reporting the results back to the coordinator.

Following, we will provide an overview of the methodology by showing how it handles the different events. These events include the building and maintaining of the IP hierarchy, the introduction or deletion of collaborating sites (that host migrating crawlers) in the system, and the delegation functionality of new sites.

7.5 The IP address hierarchy and crawlers placement

Efficient delegation of URLs to the migrating crawlers is based on an IP addresses hierarchy. We will now report on constructing and maintaining the hierarchy, and see how new and leaving migrating crawlers bind in it.

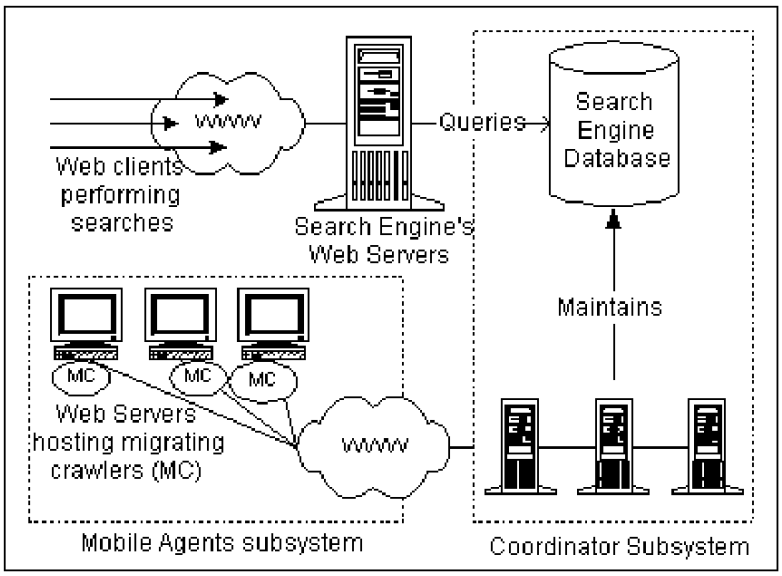

Figure 7.1: IPMicra architecture

7.5.1 Constructing the IP addresses hierarchy

The idea behind the IP addresses hierarchy is to somehow identify the various subnets and how they are linked together (i.e. they belong to the same company or a smaller subnet is enclosed in a bigger subnet). This hierarchy's construction requires the `WHOIS` data, available from the four Regional Internet Registries in bulk (one `gzipped` text file per RIR) (to be more precise, we could build the same hierarchy progressively by using the mask addresses of each IP - CIDR protocol - but due to difficulties in collecting the mask addresses, we chose to build the hierarchy with the RIR data).

Construction of the IP addresses hierarchy is straightforward and computationally light. Starting from the biggest to the smallest IP ranges (from top to down in the hierarchy), we enter the IP address ranges in the proper places in our hierarchy. These ranges are extracted from the `WHOIS` files of the RIRs. At the end of the process, we get an IP addresses hierarchy similar to the one in figure 7.2 (with more subnets of course). Each node of the hierarchy contains the IP range of the subnet and the name of the company or organization that owns the subnet.

Our experience shows that the expected maximum height of our hierarchy is 8. Furthermore, the required time for building the hierarchy is also small, and the hierarchy can be easily loaded in main memory in any average system.

7.5.2 Inserting and removing migrating crawlers from the system

The ideal approach for IPMicra is to have one migrating crawler in every subnet (as the subnets are registered in RIRs). However, this is not feasible since not all organizations

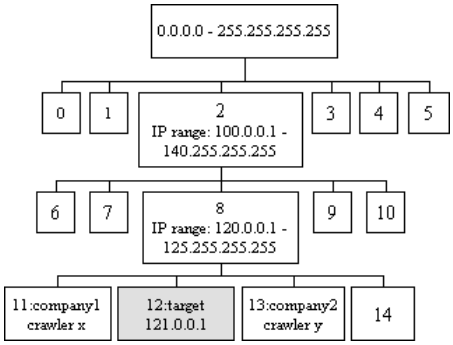

Figure 7.2: A sample IP hierarchy. Subnet 11 and subnet 13 belong to company 1 and company 2 respectively. The mentioned hierarchy can be built from the bulk `WHOIS` information from the RIRs

are willing to collaborate on such a project. So, the next best alternative is to place the crawlers in as many willing organizations as possible (usually universities and research affiliates).

After securing willing organizations, we construct and send a migrating crawler in each of these organizations (similarly with UCYMicra). Since the IP address of the machine that the crawler will run is known, we can immediately assign the subnet where the machine belongs to the new crawler. This way, the crawler gets in the hierarchy. Once set in the hierarchy it enables other load balancing algorithms (described later) to run, allowing acquisition, for balancing and performance purposes, of subnets from neighbouring migrating crawlers. Organizations canceling their collaboration with us (leaving migrating crawlers) are also handled effectively. In case of a migrating crawler leaving from the system, the algorithm runs in the coordinator to optimally re-delegate the orphan subnets.

7.5.3 Maintaining the IP address hierarchy

We do not expect that the IP hierarchy is stable over time. The current Internet infrastructure is considered one of the more dynamic and more rapidly changing structures of the world. To face this fact, we had to design a system that could be easily updated with the new data. Furthermore, it was important for us to be able to pass over to the new hierarchy all the delegations that existed in the previous one. Thus, we have developed software to automate the parsing of the `WHOIS` data from the four RIRs, and its integration into a local database for ease in searching. During testing of our software, we found that it is sufficient and inexpensive to rerun this software once a month in order to renew our IP hierarchy. Since it was computationally inexpensive, we selected to rebuild the IP hierarchy tree from scratch every time instead of maintaining the old one.

After building the hierarchy, we had to propagate all the subnet delegations that

existed in the old hierarchy tree to the new one (URL delegation process end effectively initial subnet delegation is described in section 7.6). At this point, we were handling three cases:

1. A subnet that in the old hierarchy was delegated to a migrating crawler, remained unchanged in the new hierarchy. This was the most usual case. In this case we re-delegate the subnet in the new hierarchy to the same migrating crawler.

2. A subnet (or part of it) that in the old hierarchy was delegated to a migrating crawler, is split to two or more subnets in the new hierarchy. In this case, we delegate the resulting subnets to the same migrating crawler that the original subnet was delegated to.

3. A subnet (or part of it) that in the old hierarchy was delegated to a migrating crawler, is deleted in the new hierarchy i.e. two subnets were joined to form a bigger subnet. In this case, we delegate the new subnet to the same migrating crawler that the original subnet was delegated to.

While not obvious, the last two cases are giving valid results, since changes in the subnets usually happen in low levels of the hierarchy (leafs), where we expect similar behaviour between sibling subnets.

Our experiments revealed that this algorithm was efficient for the propagation of the delegations. However, we do not expect the algorithm to be always valid. Since there are so many parameters affecting the Internet infrastructure, no algorithm can propagate the subnet delegations 100% correctly. We use this algorithm only as a best-effort heuristic, so that we do not have to re-delegate all the modified subnets each time we build the hierarchy. However, we take a provision in order to detect errors from this propagation and other errors generated from the Internet's dynamic nature. This dynamic calibration of the URL delegations is described in section 7.6.2.

7.6 The URL delegation procedure

We had to keep some things in mind while designing the delegation algorithm for the URLs. First of all, we had to make the delegation procedure as light as possible, for network and for processors. Then, since our methodology is based in the changing Internet structure, we also have to constantly verify that the delegations of the subnets to the crawlers remains near-optimal during time. Finally, we needed to alleviate any possible bottlenecks in the distributed crawlers with efficient load balancing between the crawlers.

7.6.1 Handling of new URLs

Based on the assumption that the sub-networks belonging to the same company or organization are logically (in terms of network distance - latency from other outside probing points) in the same area, we use the organization's name to delegate the different domains to the migrating crawlers. In fact, instead of delegating URLs to the distributed crawlers, we delegate subnets. We do that in a *lazy evaluation manner*, that is, we try to delegate a subnet only if we found one URL that belongs to that subnet.

To delegate a new URL, we first find the **smallest** subnet from the IP hierarchy that includes the IP of the URL, and check if that subnet is already delegated to one crawler. If so, we then try to delegate the URL to the migrating crawler that handles the subnet. If not, we do not immediately probe the subnet. Instead, we check whether there is another subnet that belongs to the same company and is already delegated to a migrating crawler (or more). Only if this search is unsuccessful, we probe the subnet with the migrating crawlers in order to find the best one to take it over. If on the other hand, we find subnets belonging to the same company (as already described, this information is saved in the hierarchy) that are already delegated to a crawler (or more), the new URL, and subsequently, the subnet that the URL belongs to, is also delegated to the same crawler (based on the assumption that the sub-networks that belong to the same organization-company are logically near, since usually companies have high-bandwidth inter-communication). That is, if one subnet handled from company X is already probed and delegated to one of the migrating crawlers, whenever we find an address residing either in the same subnet or in another subnet belonging to the same company (its IP belongs to the same company), we do not have to probe it. Instead, we delegate it to the same migrating crawler that has taken the first subnet. Furthermore, if there are more than one subnets of the same company delegated to different crawlers (this is possible in our algorithm in order to delegate sites belonging to international companies, and for load balancing), then the new subnet is probed from these crawlers and delegated to the fastest. And since probing is done sequentially, we stop as soon as we find a crawler that satisfies the probing threshold.

From our method we exclude the unary subnets (subnets that include one address only) since delegation of these subnets to crawlers would result in minimal benefits. In the case where URLs belong to one of these unary subnets, these subnets are ignored and their immediate parent subnets (the subnets that include the unary subnet) are used.

Subnet probing is done by probing a URL that belongs to the subnet (since we probe the subnets only when we found a URL, we must have already found a URL). However, not all the crawlers need to probe all the subnets. We can avoid probing from most of the crawlers, even for a completely new subnet. Subnets form a tree-like

hierarchy. Since the IP addresses are assigned from ICANN to RIRs and from RIRs to LIRs (Local Internet Registries), and then to organizations, usually in batch mode (as IP ranges, called subnets), it is logical for us to follow the hierarchy from bottom to up (subnets from small to big), each time trying to find the most suitable crawler to take the subnet. Thus, when probing needs to be done for a subnet, we first discover the parent subnet (the subnet that includes our target subnet), and find all the subnets included in the parent subnet (can be seen as siblings of our target subnet). Then, for all the sibling subnets that are already delegated, we sequentially ask their migrating crawlers, and the migrating crawlers of their children subnets to probe the target subnet, and if we find any of them that has probing time less than a specific (probing) threshold, we delegate the target subnet to that crawler and stop probing. If none of the migrating crawlers satisfies our probing threshold, our search continues to higher levels of the subnets tree. In the extreme case that none of the crawlers satisfies the probing threshold, the subnet is delegated independent of the probing threshold, to the crawler with the lower probing result.

The main logic of the algorithm described earlier (executed in the coordinator) is presented in the following pseudo-code:

```
for any newly discovered URL u {
  subnet s = smallestNonUnary(u);
  if (IsDelegated(s)){ // the subnet is delegated
    delegate u, s to the same migrating crawler;
    next u;
  }
  elseif (sameCompanySubnetDelegated(s.companyName)){
  // a subnet of the same company is delegated
    mc = migrating crawler that has the other subnet;
    mc.delegate(u, s) //the url and the subnet
  } else {
    while (s not delegated) {
      s = s.parent;
      if (IsDelegated(s)) { // check the parent
        mc = the migrating crawler that has subnet s;
        time = mc.probe(u);
        if (time<threshold)
          mc.delegate(u, s); //the url and the subnet
      }

      for every child of s until u is delegated {
        sch = s.child
```

```
        if (IsDelegated(sch)) { // check the child
          mc = migrating crawler that has subnet sch;
          time = mc.probe(u);
          if (time<threshold)
            mc.delegate(u, s)
        }
        if (allAvailableCrawlersProbed)
          delegate the subnet to the fastest crawler
      }
    }
  }
}
```

Some companies may have multiple mirrors of their main web-sites distributed over the world (in different locations) and use a DNS-based load balancing or a commercial solution like the ones offered from CDN's e.g. Akamai. However, this does not affect our algorithm, since we delegate IP addresses (more exactly, IP subnets), and not web sites. So, the fact that there can be multiple mirrors and multiple IP addresses of the same domain name does not create any trouble. We just select one of these IP addresses, and delegate it to the most suitable crawler. If for any reason the selected mirror ceases working in the future, we then retry the delegation for an alternative IP address. The described approach may not always offer the optimal solution, since another pair of a migrating crawler and a mirror may have better results, but it has good results, and testing all the combinations is not something light for the network.

Probing: Probing of subnets is inexpensive. However, it was important for us to respect the communicational resources of the machines that were hosting the migrating crawlers. Therefore we enabled subnet probing only during the time that the migrating crawler is allowed to crawl (the low-usage hours). Therefore, any subnet probing request remains in the coordinator until the timespan that the destination crawler is allowed to work arrives. Then the site is probed. This has the extra advantage that each site is probed from each crawler near the time that it would be actually crawled later if it was delegated in that crawler. This kind of probing takes the expected workload of the server to be crawled in the future crawling time under consideration, thus avoids overloading of the server in peak hours. The extra time needed for the probing to be completed is insignificant.

Probing results collected from the various crawlers are returned in the coordinator, who needs them to perform the location aware delegation, and are stored in the central database for future use. This practice saves us from an important number of probes, since we may recall a probing result from the database (if exists) instead of trying to

find it again. Invalidated probing results are easily located and replaced, with a dynamic calibration mechanism described later. Furthermore, this probing database enables the search engine administrators to define the appropriate probing threshold.

Example: In order to clarify the delegation methodology, we include an example. The example references the hierarchy presented in figure 7.2

Subnet 2 in figure 7.2 has an IP range from 100.0.0.1 to 140.255.255.255. Subnet 8 is included in subnet 2 with an IP range from 120.0.0.1 to 125.255.255.255. Subnet 12 is a unary subnet for IP 121.0.0.1. The mentioned hierarchy can be built from the bulk `WHOIS` information from the RIRs. The scenario includes probing for a URL that resides to IP 121.0.0.1. Doing a query in our IP addresses hierarchy, we discover that the smallest subnet including our target IP is subnet 12, which however is unary. Thus, according to our algorithm, we ignore subnet 12, and use subnet 8 instead. Subnet 8 is not delegated in any crawler, so we check to see if any other subnet belonging to the same company is already delegated to any crawler. Supposed that no other subnet of the same company is delegated (organization name is stored in every node in the hierarchy), we continue by checking for neighboring subnets that are delegated. Looking again in our hierarchy, we discover that while subnet 8 is not delegated to any crawler yet, subnets 11 and 13 (its children) are delegated to two different crawlers, x and y respectively. Therefore, we ask these two crawlers to probe the new subnet. If probing in either of the two crawlers' results in time less than the probing threshold, we delegate the new subnet to that crawler, or else we proceed to higher levels of hierarchy. However, since in this scenario, subnet 12 is a unary subnet, we delegate both subnets 8 and 12 to the faster crawler. Since the subnets 11 and 13 are already delegated and are lower in the hierarchy than subnet 8, this does not affect them (their delegation supersedes the delegation of their father). Subnet 14, which is not yet delegated, stays un-delegated. If we need to delegate it in the future, we run the same algorithm until we find some crawler satisfying the probing threshold.

7.6.2 Dynamic calibration of URL delegations

The assumption that sub-networks belonging to the same company are logically near (in terms of network latency) may, in rare cases, be wrong, e.g. in international geographically distributed companies that do not provide fast interconnections between their offices. Furthermore, the probing results stored in the database as well as the subnet delegations may get invalid during time, since they are based in the changing Internet infrastructure (thankfully, these changes are not expected to happen in high frequency, since they mostly depend on very expensive commercial agreements. Our experience showed that the probing results were not significantly changed in a testing period of

three months).

An efficient algorithm should detect these changes in order to maintain an efficient delegation of subnets to migrating crawlers. For these reasons, we perform some additional counts during the crawling function, which efficiently detect changes that should cause re-delegation of some subnets. More specifically, each migrating crawler in the IPMicra extension counts the required time for every `HTTP/GET` request it issues for every web page. The new time is compared with the previous `HTTP/GET` times for the same web page, stored locally in the crawlers' memory. If the new time is sufficiently larger (a threshold defined from the search engine administrator) than the time demanded for the previous downloads of the same page, and if this repeats for more than one time continuously, then the subnet is re-delegated, so that a more suitable crawler is found. The same happens with every `HTTP/HEAD` command, which is often used from our crawlers in order to discover whether a particular page has changed.

Incorporating this extra task during web crawling, we manage to keep our delegations up to date and to efficiently detect any negative changes in the Internet infrastructure with respect to the existing delegations. Applying the dynamic calibration algorithm does not introduce any more load in the Internet infrastructure, since we only use information already available, produced from the standard commands issued for the web crawling task. Furthermore, the extra computational load added from the dynamic calibration algorithm is negligible.

7.6.3 Bottleneck elimination and load balancing

The previously described delegation of subnets to crawlers can create bottlenecks to crawlers that are connected with high-bandwidth links to the Internet or crawlers that are in very crowded (from web-sites) areas. Furthermore, the organizations hosting the crawlers may choose not to let the crawlers take advantage of their full free bandwidth, even in the low-usage hours that the crawling is done (as with the UCYMicra approach, IPMicra too crawls only in low-usage hours i.e. from midnight to six in the morning), for their own personal reasons. For these reasons, we enhanced our algorithm to take these conditions into consideration. Each crawler has a maximum capacity, the size of the assigned web-pages that the crawler has to check each day. In the case where one crawler is found to be the best for one new subnet, but the inclusion of this subnet to the crawler's workload violates the crawler's capacity, then measures are taken toward one of the following: (a) either the overloaded crawler takes the new subnet and releases one of the old ones, which is then taken over from some other crawler, or (b) the new subnet is delegated to the second-best crawler.

We looked into several approaches to ensure that our suggestion is scalable. We were

able to find many suitable approaches for this task. The problem could be translated and solved as a standard linear programming problem, or, we could follow a test-all-cases approach, testing all the possible combinations in order to find the best. Furthermore, we could use a simpler heuristic that finds the best URL to re-delegate by comparing the next-best time for all the URLs in the overloaded crawler. All the approaches target in minimizing the global cost, which is, in our case, defined as the sum of the time that takes each of the crawler to download the web pages assigned to it $\sum_{\forall i,j} P_{ij}$, where i=page, j=crawler, and

$$P_{ij} = \begin{cases} \text{time required to download i from j,} & \text{if i delegated to j} \\ 0, & \text{otherwise} \end{cases}$$

For the purpose of this work, we developed another heuristic based on the variance of the logical distance of each web-page (identified from the probing result) from all the migrating crawlers that probed the web page. Intuitively, small probing time variance implies that most of the probed crawlers have similar probing results. Thus, we expect to be easily able to find a near optimal crawler to take over a page. The execution of the load balancing algorithm is taking place in the coordinator. The coordinator uses the stored (in the database) probing results for all the subnets to select the subnet that can be removed from the overloaded crawler. This is the subnet with the smallest variance in its probing results. The crawler takes this subnet and checks if the next-best crawler for it (if any), according to the probing results, is available and can accept the subnet (i.e. will not get overloaded). If so, the subnet is re-delegated to this crawler. If not, we retry the approach with the next best subnet, until we find a subnet that can be efficiently delegated to a new crawler.

We found this heuristic to perform well. We selected this heuristic instead of a linear programming (which would be more effective) and the test-all-cases solution strictly because of the ease in implementation. Furthermore, the heuristic was selected over the heuristic based on next-best time, because it was expected to scale more gracefully, and function better over time. Our tests revealed that the heuristic was able to make the best load balancing decision in more than the 2/3 of the cases. Furthermore, in all the rest cases, the heuristic was able to find an acceptable solution. Unfortunately, due to space limitations we cannot present analytical results of our experiment here. While satisfied with this heuristic, part of our ongoing work is to apply and evaluate other load balancing algorithms.

7.7 The evolutionary nature of IPMicra

IPMicra optimizes itself during time. As already noted, IPMicra performs a dynamic calibration of the URL delegations in order to discover problematic delegations as soon as they become non-optimal. Thus, the delegations in the IP address hierarchy are always kept near to optimal. Furthermore, as the IPMicra system gets 'trained' with more data (more migrating crawlers and more web sites), it requires less and less probes to perform an optimal delegation. More specifically, while adding the subnet delegations in the IPMicra hierarchy we increase the possibility of a new URL to reside in an already delegated subnet (thus, assigned to a crawler without probing), or, very near a delegated subnet in the IP hierarchy (which makes it easier for the algorithm to find the best crawler).

The previous argument can be verified from the experimental data that was collected during the evaluation of IPMicra. As presented in section 8.6, the last 50 URLs had less average probes per URL from the first 300 URLs, in order to succeed a successful delegation. This is mainly because the hierarchy was getting optimized-trained with the subnets' delegations.

In order to clearly visualize the subsequences of training the IPMicra hierarchy, we performed another experiment with IPMicra, this time without performing any mass delegation of the URLs. More exactly, we got a clean (without any delegations) IPMicra system and only placed the agents in the hierarchy. Then, we started delegating 1000 sites, always using the IPMicra algorithm, and measured the required probes for every site. The results (for three different thresholds) are summarized in figures 7.3, 7.4, and 7.5. It is very important to see the slope of the trend line (a least square regression line) in these figures (the dashed line) which makes it clear that the required probes are continuously reduced during time. This practically means that our method can scale in pace with the web's increasing size and a significant number of collaborating migrating crawlers. In fact, IPMicra actually gets more optimized and requires less probes while adding more sites and more crawlers.

Similarly, we expect that adding new migrating crawlers in the hierarchy (new collaborating organizations) has the same positive effects in our algorithm. More specifically, not only there are more available web crawlers to handle the crawling load, but the discovery of the optimal crawler for the new URLs gets easier (with less probes) for the same reasons.

Based in our experience with the IPMicra project, we argue that IPMicra can reach a status where it needs not more than two probes for a new URL very easily, with a reasonable number of collaborating organizations (i.e. a community of 100 organizations) and without requiring long 'training'. While, with only twelve collaborating sites and with

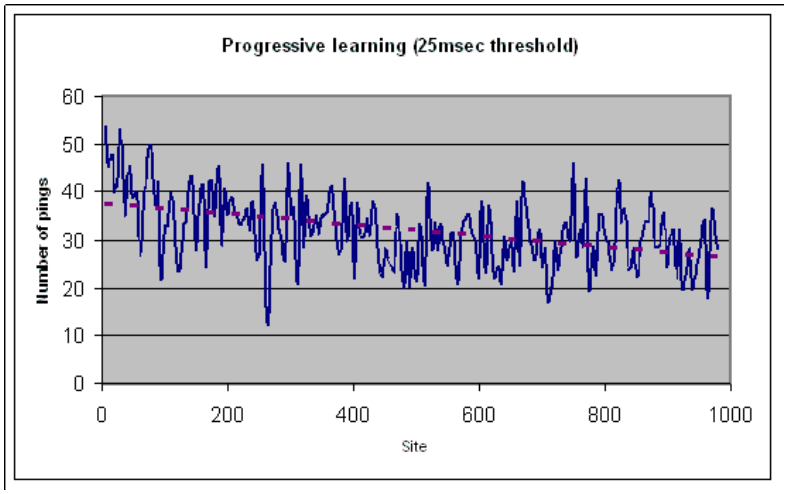

Figure 7.3: Progressive learning (threshold=25)

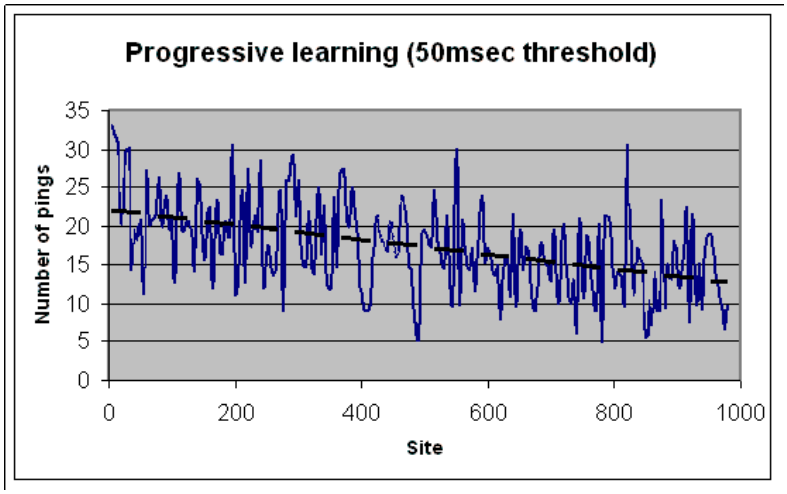

Figure 7.4: Progressive learning (threshold=50)

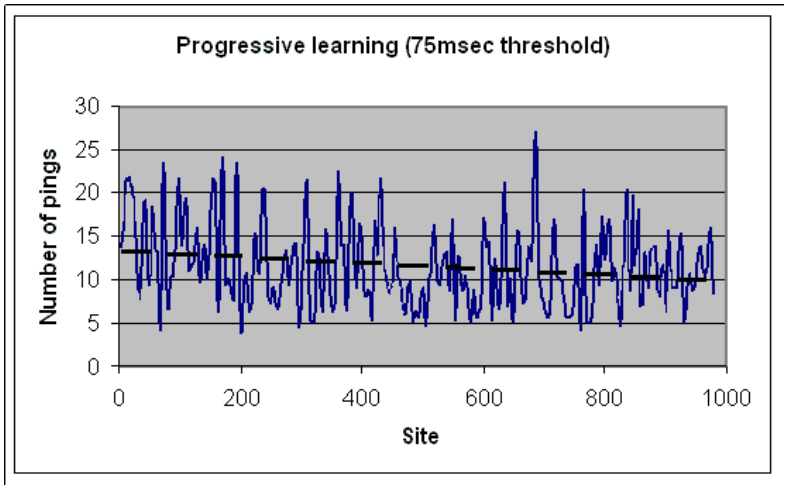

Figure 7.5: Progressive learning (threshold=75)

a small number of URLs and subnets delegated, IPMicra manages to outperform its opponents, we expect the new approach to perform miracles in a larger set i.e. in covering the complete Internet, which however we could not test due to practical limitations.

7.8 Evaluation of IPMicra

IPMicra, as a distributed crawling approach, manages to outperform centralized crawling. Moreover, IPMicra compared to other distributed web crawling approaches, requires one order of magnitude less time for crawling the same set of web pages.

IPMicra and location unaware distributed crawling: We compared IPMicra with a location unaware distributed crawling approach. While UCYMicra was a successful distributed crawling approach, the original UCYMicra suggestion was not targeting in having the migrating crawlers crawl the whole Internet, but only the pages inside the LAN of the crawlers. For this reason (as described in 5.2.2), we enhanced the original concept of UCYMicra by allowing the migrating crawlers to crawl sites outside their LAN. The sites were randomly assigned to the available migrating crawlers, without taking the location of the crawlers under any consideration. Our experiments revealed that IPMicra requires an order of magnitude less time to crawl the same set of data (with the same number of available crawlers, in the same hosts). The advantages of IPMicra are even more visible when comparing with the worst-case scenario of the enhanced UCYMicra (the case where each URL is delegated to the worst web crawler). Detailed results of our experiments are presented in 8.6.

IPMicra and centralized crawling: IPMicra is a distributed crawling approach, and, as such, its comparison with a centralized crawling approach is not at all trivial. A distributed web crawler has important advantages compared to a centralized one, since

it distributes the downloading and the processing task to the available hosts, this way alleviating the network and processing bottlenecks from the search engine site. Furthermore, distributed crawling reduces the need for network resources from the search engine site significantly for various reasons. A traditional crawler on the other hand, does not have the extra overhead of collecting the produced data from the distributed collaborating sites, which is the case of the distributed crawler.

The most important bottleneck in the traditional web crawlers is the network bottleneck, occurring at the search engine site, while web crawling is performed. Namely, the important comparison between the two approaches would be the comparison of the time required to transmit data via the search engine's network, and the size of the transmitted data over that network path, as this was documented in [35, 36]. In the traditional crawling approach, this data includes the HTTP/GET requests and their results, while in IPMicra, this data includes the compressed and processed web documents, transmitted from the migrating crawlers to the search engine coordinator (as data carriers), and some additional control messages of negligible size (i.e. probing requests and responses). Having these in mind, the IPMicra approach outperforms the traditional crawling approach by requiring to send an order of magnitude less data to the coordinator, for the following reasons:

1. The data from the migrating crawlers is processed and compressed before transmission to the coordinator. Thus, this data (as in UCYMicra [35, 36]) is at least one order of magnitude smaller than the complete web documents that contains.

2. The data transmitted from the migrating crawlers is transmitted in a batch method. Thus, it has less connection delays and overheads.

3. In the distributed crawling approach (IPMicra), not all the changed web pages need to be transfered back to the coordinator for integration in the database. As documented in [14], not all the pages that appear to have changes (with the conditional HTTP/GET or the LAST_UPDATE date in the HTTP/HEAD) are changed. Some of these pages have negligible, trivial changes (e.g. a page counter) that need not to be integrated in the database at all. A traditional crawler could not identify these pages without downloading them completely. IPMicra on the other hand is able to detect these changes, and avoid to transmit them back to the coordinator.

4. The new control messages introduced in IPMicra are of negligible size, and are not responsible for creating any network bottleneck in the search engine site.

Furthermore, IPMicra also alleviates the processing bottleneck which often occurs in the traditional crawling approach, by distributing the processing task to the available

migrating crawlers. The migrating crawlers do not face any processing or network bottleneck, since, in a non-experimental setup, they are expected to be many (Grub, a similar distributed crawling approach, now has more than a thousand users, while the users of SETI are more than a million).

7.9 IPMicra compared to HiMicra

In this work, we presented two heuristics for making a location-aware delegation to the available migrating crawlers: HiMicra and IPMicra. We set a number of experiments for the two approaches, in order to evaluate their efficiency (how many optimal delegations they succeed and how close to optimal are the rest of the delegations) and their overhead (how many pings they need to make the delegations). We also needed to find out whether the results get improved during time.

Evaluating the experimental results we can now say that both the approaches have significantly less overhead from the brute-force approach, which requires pinging of all the URLs from all the crawlers. However, we expect that IPMicra will perform better than HiMicra in the real Internet due to IPMicra's evolutionary nature. IPMicra optimizes the IP hierarchy during time, thus, requires less and less probes (average) for each new URL. HiMicra on the other hand does not have such an evolutionary nature. HiMicra does not use older delegations to delegate the new URLs.

Some of the experimental results (when testing HiMicra with the generated data) show that HiMicra can require less probes to perform a delegation from the IPMicra approach. However, the delegation produced from HiMicra is significantly less optimal than the IPMicra delegation. Furthermore, when testing HiMicra with the artificial data set, we introduce 50 crawlers in our experiment, while, for practical reasons, we were able to run IPMicra with a maximum of 12 crawlers. We expect that the IPMicra approach, when tested with the same number of migrating crawlers and the same number of sites with real data collected from the Internet, will have significantly better results than the HiMicra approach.

7.10 Conclusions from the IPMicra system

In this chapter, we proposed IPMicra, an extension of UCYMicra, which targets on offering efficient load balancing and a near-to-optimal delegation of URLs to migrating crawlers. The motivating power behind IPMicra is an IP address hierarchy tree, which we build using information from the four Regional Internet Registries.

The IPMicra approach manages to outperform any other distributed crawling approach, by requiring significantly less time (one factor of magnitude) to download the

same set of web pages. The importance in IPMicra is that this is done with a negligible probing overhead. This overhead is reduced while the IPMicra hierarchy gets 'trained' during time, and is expected to be minimal (i.e. an average of less than 2 probes per URL) in a commercial setting with many available highly distributed migrating crawlers and an optimized IP hierarchy.

We consider this work of significant importance due to the imperative need for a scalable web crawler, that will be able to catch up to the expanding and rapidly changing web. The IPMicra proposal can be the solution to the web indexing problem faced from all the search engines nowadays. Furthermore, while the location aware infrastructure developed in this work is powered from UCYMicra, it can be applied (as a framework) to any other (fully or partially) distributed web crawler. The framework can even be applied in existing commercial approaches, like the Google Search Appliance or Grub.

Moreover, the location-aware infrastructure developed in this work can be used as a framework to facilitate several optimizations itself, for generic distributed applications in the Internet. For example, this framework could efficiently enhance the load balancing schemes used from content delivery networks, such as Akamai.

Chapter 8

Evaluation scenarios and analytical results

8.1 Introduction

In this chapter we present the evaluation results of location aware web crawling, HiMicra, and IPMicra. The evaluation results show that location aware web crawling manages to downgrade the time needed for downloading of the same data to as much as an order of magnitude. The same applies for the brute force location aware web crawler, which uses `ping` or `HTTP/HEAD` messages to estimate the logical distance.

From the evaluation it is also clear that HiMicra and IPMicra manage to make good estimations for the location awareness, without the need for excessive probes. We also show that the two approaches outperform not only the traditional crawler, but even the more advanced UCYMicra (distributed) crawler.

In order to get a better insight of our evaluation, we performed each experiment presented below for a number of times, and then calculated the averages. Furthermore, each time we experimented not only with the subset of sites but also with the time of the day the experiment would run, and whether the experiment would run concurrently in all the sites. In short, the results of all the experiments favored location aware web crawling, and especially the IPMicra approach. The results are presented in more detail below.

8.2 Evaluation of location aware web crawling

Web crawling can be divided in three distinct functions, (a) the download of the page, (b) the processing of the HTML text, and (c) the integration of the results in the database. Location aware web crawling, compared to distributed location-unaware web crawling (UCYMicra), was expected to reduce the time needed for the first function, the downloading of all the pages. The processing of the HTML text and the integration of the results in the database are not affected from the location aware extension, so, we do not expect them to change.

In order to verify our arguments we ran the following experiment: The experiment demanded crawling of 1000 static web pages (different domains) from four distinct locations (physically and logically far away from each other) and timing the download function for each page from each crawler. At some of the times the crawling was done concurrently from all the crawlers and at some other times asynchronously - in low-use local hours for each crawler. To get more clear evaluation, we ignored the processing time, and the time for integration in the database, since these times were not affected from the location aware approach. We only count the required time to download all the web pages from the crawlers. The downloading function in each crawler was done sequentially, meaning that, at each moment, there was a maximum of one active HTTP connection at each crawler for downloading a web page. This way we were trying to keep our results independent of the available network bandwidth, so as to more accurately measure the effects of location aware web crawling. We had to enforce this limitation since each crawler was running in different network, and having different bandwidth to use.

We then used the timing results that were collected during the experiment in order to simulate and compare the two approaches. In order to get the setup for a location aware web crawling, we delegated each URL to the crawler that needed the less time for downloading it. The setup for the enhanced UCYMicra(as described in 5.2.2), representing distributed location unaware web crawling, was found by performing a random distribution of the URLs to the four migrating crawlers. The experiment was repeated multiple times, with different URLs each time, for more representative results.

Our evaluation had very encouraging results. Location aware web crawling outperformed standard distributed crawling (represented from the enhanced UCYMicra, which is even better than the earlier distributed and centralized web crawlers) since it managed to significantly reduce the time needed to download the web pages. More specifically, location aware web crawling required **one order of magnitude less time** for downloading the same set of pages than the UCYMicra approach. The results of location aware web crawling were even more impressive compared with the worst case

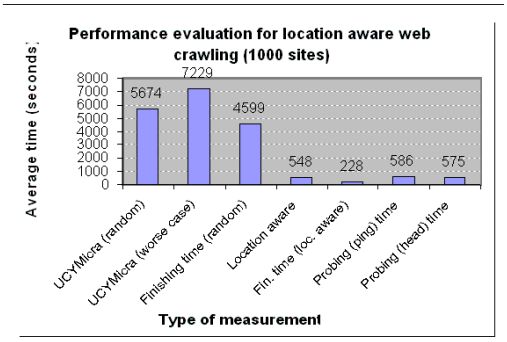

Figure 8.1: Performance evaluation for location aware web crawling

scenario of UCYMicra. The results of this experiment are summarized in 8.1.

8.3 Evaluation of the probing metrics

It was straightforward to evaluate the suitability of the two proposed probing metrics for estimating the logical distance between two Internet nodes. To do so, we performed the following experiment: We constructed a slightly modified version of the migrating crawler suggested for UCYMicra, which was able to measure the time required to download a web document. The crawler was also enabled to probe a foreign host (with `HTTP/HEAD` and `ping`). We then launched four modified migrating crawlers to four different places (geographically and logically far away from each other). Finally, we fed the four crawlers seedlists with a list of 1000 URLs (randomly selected from our database). The crawlers not only downloaded the URLs and measured the required time to download each page, but also probed the machines hosting the URLs, and saved all the measures in a text file. We then collected and analyzed the four text files, in order to reveal if there is any relation between the probing results and the time required to download a web page. In order to have better results, we repeated the experiment four times, each time with a different set of URLs, and at different time of day.

The experiment had very encouraging results. The `HTTP/HEAD` and `ping` metrics were both proved very suitable for the task of estimating the logical distance between two Internet nodes. In more than 90% of the cases, the crawler that had the shortest probing result for a page, was also the crawler that could download that page the fastest. More specifically, while using `ping` for probing, we were able to make 908 (average) out of 1000 correct predictions about which crawler was the best (the fastest) for each page. Furthermore, while using `HTTP/HEAD`, we were able to make 925 (average) out of 1000 correct predictions. In these numbers, we included the cases that, for some reason (e.g. temporary network downtime in the target machine), one or more of the crawlers failed to probe a URL. In these cases, we were predicting the nearest crawler

based on the available probing results, or, at the worst case (where all the crawlers failed) randomly.

These results practically meant that we found two efficient and light methods for estimating the most near web crawler to an arbitrary Internet node. The methods, as evaluated experimentally, were giving almost completely correct results, and by generating importantly less load than by timing the `HTTP/GET` command.

8.4 Evaluation of the brute force web crawler

Using the same setup, we also experimented to find out the suitability of the two suggested probing metrics (`ping` and `HTTP/HEAD`) in order to get a better insight for the brute force location aware web crawler, suggested in section 5.3. More to the point, the crawlers in the previously described experimental setup also had a probing functionality enabled. Probing was either done with an inexpensive `ICMP PING` message or by timing an `HTTP/HEAD` request. Thus, the executions of the previous experiment also produced some probing results, which we could use to simulate the brute-force approach described earlier. However, in the probing results we had some cases where at least one of the crawlers failed to probe a particular URL. This failure was either because of a temporary network or web server malfunction, or, because of network restrictions i.e. an intermediary firewall was blocking `PING` messages. These URLs were detected and counted. Then, these URLs were delegated by using the available probing results (or, at the worse case where there was no available probing result, randomly). Finally, for the rest URLs (the URLs that were probed normally from all crawlers), instead of simulating the most efficient location aware delegation, this time, we made the delegation based on each of the two probing metrics (`ping` and the time required for an `HTTP/HEAD`).

The results of these two experiments are summarized in figure 8.1 where we present the following parameters:

Column 1 - UCYMicra[1] (random) The time required for downloading with UCYMicra when we were randomly delegating the sites to the available crawlers. This time was the sum of all the downloads from all the crawlers (ignoring that the crawlers were running in parallel).

Column 2 - UCYMicra (worse case) The maximum possible time required for downloading with UCYMicra. This field was calculated by making the worse possible delegations for each URL.

Column 3 - Finishing time (Random) Considering that all the crawlers were downloading in parallel, this is the time that took for all the crawlers to finish in the UCYMicra version, since the downloading started.

Column 4 - Location aware The time required for downloading in our distributed crawler when we were having a perfect location aware delegation. This time was the sum of all the downloads from all the crawlers (ignoring that the crawlers were running in parallel).

Column 5 - Finishing time (location aware) Considering that all the crawlers were downloading in parallel, this is the time that took for all the crawlers to finish in the location aware version, since the downloading started.

Column 6 - Probing (ping) time The time required for downloading with brute force location aware methodology when we were using `ping` for probing. This was the sum of all the downloads from all the crawlers (ignoring that the crawlers were running in parallel).

Column 7 - Probing (head) time The time required for downloading with brute force location aware methodology when we were timing the `HTTP/HEAD` command for probing. This time was the sum of all the downloads from all the crawlers (ignoring that the crawlers were running in parallel).

We then compared the resulting delegation from the brute force approach with the optimal delegation. The results of this comparison were very positive. While we were not able to always make the most optimal delegation, this was the case in the overwhelming majority of the URLs. The probe-based delegation (both the approaches - with `ping` and `HTTP/HEAD`) correctly identified the best crawler for most of the URLs. When using `ping` to probe the remote sites from each crawler, we managed to correctly delegate 908 URLs (average), while when probing with `HTTP/HEAD` we managed to correctly delegate 925 URLs (average). This resulted to an average of 92 suboptimal delegations for `ping` and 75 suboptimal delegations for `HTTP/HEAD`. As already mentioned, in these numbers we included the cases where some or all the crawlers failed to probe the site. The delegation in this case was done either from the remaining probing results, or randomly. In our experiment, when using `ping` for probing we had an average of 76 such failures, while `HTTP/HEAD` had only 12 such failures.

Concluding, it is important to note that, from the evaluation results and especially Columns 4, 6, and 7, it is clear that the two probing alternatives we suggested (`ping` and `HTTP/HEAD`) can easily and effectively be used toward a location aware web crawling system.

8.5 Evaluation of HiMicra

For practical reasons, we were not able to correctly evaluate the HiMicra system. Such an evaluation would require a big number of collaborating organizations (that would

host us a migrating crawler) and a big number of testing URLs. However, we were able to do some preliminary tests with 12 migrating crawlers, which we will present now. We do not argue that the preliminary tests fully represent the algorithm, since the algorithm was designed for bigger testing sets (more migrating crawlers, bigger hierarchies). In fact, we expect the algorithm to have significantly better results in the real Internet, with more migrating crawlers. Also, due to the limited number of migrating crawlers we could not even try different variations of the algorithm (i.e. try to run the algorithm with a branching factor set to 3).

For our preliminary experiments, we managed to collect probing measures from 12 collaborating organizations (which were kind enough to run a script for us, that sequentially probed 1000 URLs, and propagated the results back to us). Then, we used the probing results from the 12 organizations in order to simulate the HiMicra approach and evaluate its performance.

More precisely, we collected probing measures (for practical reasons we were probing with the `ping` command, although we expect that probing with `HTTP/HEAD` will have very similar results, according to section 8.3) for 1000 randomly selected domain names from 12 affiliated universities and companies, which were widely spread over the world(U.S.A, Europe, Australia). The probings were each time collected in low use hours (local time) in order to avoid creating any trouble and get more representative results. Then, we ran a number of simulations of the HiMicra approach, each time selecting a different subset of the 1000 URLs to construct the hierarchy, and a different subset of the 1000 URLs to execute HiMicra. We also experimented with different probing thresholds. Each time, we measured the average number of the required probes for each delegation in the test set, the percentage of the optimal delegations, and the average probing difference for the non optimal delegations (the average difference between the optimal and the suggested delegation, calculated with $\sum_{s:site} \frac{P(opt,s)-P(delegated,s)}{n}$, where $P(opt, s)$ is the optimal probing result for site s, $P(delegated, s)$ is the probing result of the suggested delegation for site s, and n is the number of the non-optimal delegations).

The preliminary results were very promising, considering the small number of the migrating crawlers. We managed to significantly reduce the required number of probes for each delegation (compared to the brute-force approach) and at the same time find a near-to-optimal delegation. More precisely, with probing threshold set to 25msec and with different number of sites used for building the hierarchy (250, 500, 750), we managed to find an average of 89% optimal delegations with 78% of the probes that were demanded in the brute-force approach. At this setup, the probing time difference (for the non-optimal delegations) was 5msec. For a probing threshold set to 50msec, the new results were 62% optimal delegations with 55% of the probes (again, compared with the brute force approach), while the probing difference was 17msec. Finally, with

Threshold	Optimal(%)	Suboptimal(%)	Probing Time Difference	Required Probing(%)
25msec	89%	11%	4.8msec	78%
50msec	62%	38%	17msec	55%
75msec	43%	57%	22.9msec	42%

Table 8.1: Results of HiMicra with real data collected from the Internet

a probing threshold set to 75msec, the optimal delegations were 43% and the number of probes 42% while the probing difference was only 23msec. The results are also summarized in table 8.1.

As already noted, since we had an extremely small set of collaborating organizations we were unable to test our approach with different branching factors (i.e. set the maximum number of children to more than 2). For example, setting a maximum number of children to 3 would result in only two levels in our hierarchy (apart from the coordinator), in which case the hierarchy would hardly serve its purpose of making a delegation with less probes.

During our simulations we also experimented with various subsets of training sites (sites used for the construction of the hierarchy). As expected, due to the small number of URLs that could be used for building the hierarchy (we only had 1000 probed sites), the results when increasing/decreasing the URLs used for training were not significantly differing. However, we expect that with a bigger training subset i.e. 10000 URLs (which would be the case in the real Internet) the algorithm will be able to produce a better hierarchy (more optimal delegations and less probes).

While at this experiment we had to reuse some of the URLs that were used at the process of building our hierarchy, since we only had 1000 URLs, this was not interfering with our results. The process of building the hierarchy in HiMicra does not result at any delegations. Furthermore, the probes we were using in the hierarchy building process were not available to the delegation task. Instead, the coordinator was counting every probe that was required, independent if this probe was also used during the construction of the hierarchy.

The above results and our experiments did not include the actual execution (i.e. data crawling) of the suggested delegation for practical reasons. Running such a function in the twelve foreign hosts could be characterized as network attack, and would also take significant time and use important resources, that the collaborating networks could not spare. However, based in the previous experiments (section 8.3 and section 8.2), during which we showed direct correlation of probing and downloading time, we argue that the downloading time for the HiMicra delegation would be very near to the optimal downloading time. Thus, we expect the downloading time to be one order of magnitude

Optimal delegations (%)	Average time difference in the non-optimal delegations	Required probes(%)	Number of crawlers
90%	18msec	68%	10
63%	19msec	37%	25
50%	20msec	27%	40
46%	21msec	18%	50

Table 8.2: Influence of the number of crawlers in the HiMicra methodology

less compared to the average downloading time with a random assignment of URLs to the available crawlers.

In order to get a better insight of our methodology, we simulated HiMicra with a larger set of generated data. More precisely, we generated a data set with 50 crawlers and 2000 URLs. The generated probing times for the 50 crawlers were clustered around 50 gaussian distributions of varying mean value and deviation. While the data used in this experiment was not necessarily representing the Internet real structure, with this test, we could easily test the scalability of the method, that is, how the method reacts when the number of crawlers is increased. We repeated the experiment with 10, 25 and 40 crawlers, and compared the results.

The results of the last experiment (with the generated data) revealed that, while the number of crawlers increases, the percentage of the required probes compared to the respective probes needed in the brute-force approach is importantly decreased. For example, with the same threshold (25msec), and with the same subset of sites devoted to the building of the hierarchy (250 sites), when having 10 crawlers we managed to reduce the number of the required probes for the delegation to 66% (compared to the brute-force approach). With 25 crawlers, the required probes were reduced to 39%, and with 40 crawlers they were reduced to 27%. Finally, with 50 crawlers, we only had to make 19% (less than 1/5)of the probes that a brute-force approach would require. These results were very similar in all our experimental setups (with different probing threshold and different size of the subset of URLs used in building the hierarchy). In table 8.2 we summarize our results with a probing threshold set to 50msec and 500 sites used for training. We also noted that the optimal delegations in our experiments were decreasing while the number of crawlers was increasing. Nevertheless, this does not reduce the value of the methodology for the following reasons: (a) the suboptimal delegations were very near to the optimal ones, and (b) this may not be the case with real data, collected from the Internet.

In this experimental setup, the number of the sites used for building the hierarchy was not affecting the results significantly. While we had the opportunity to train the hierarchy with a significant number of sites, the probing results for these sites were

all generated from gaussian distributions. Thus, when varying the quantity of the sites used for building the hierarchy, the results did not significantly change. In fact, using 250 sites for building the hierarchy was sufficient for capturing the artificial Internet structure since it resulted in very similar results with the ones we had when using 1750 sites. For instance, varying only at the number of sites used for building the hierarchy, with the training threshold set to 25msec and 10 available crawlers, we got the following results:

# of sites used for training	Required probes (%)	Optimal delegations (%)
250	66%	92%
500	68%	89%
750	67%	90%
1000	69%	89%
1250	67%	91%
1500	68%	90%
1750	66%	92%

We do not argue that the results of the experiment with the generated data stand in the real Internet. However, the experiment reveals the importance of the number of the crawlers. Under the same setup (with the same set of sites) we can see that HiMicra saves more probes as the number of crawlers increases. We expect this to be valid also in the real Internet, and this is the exact importance of this experiment.

8.6 Evaluation of IPMicra

In a previous section we evaluated location aware web crawling and confirmed that it can actually reduce the downloading time to one order of magnitude. However, the brute force location aware web crawling approach required an extensive number of probes, making the method difficult to scale. For this reason we suggested the IPMicra approach, which used data from the four RIRs to construct an IP addresses hierarchy and use it to efficiently delegate each URL to the most near crawler. The evaluation of the IPMicra methodology was not trivial and in order to correctly evaluate it, we should first run it with a mass amount of real data. This is necessary since the assignment of subnets (and thus URLs) to the various crawlers is dynamically adaptable (see section 7.6) and an optimal placement with minimal probing is achieved over time. In fact, in a real environment it should be done this way. Then, we should introduce a number of completely new URLs in the methodology and measure the following parameters: (a) the success rate (the correct delegations), (b) the required number of probes, (c) the time

required to execute the delegation (to download the URLs from the crawlers that have been delegated to), and, (d) the time required to execute the optimal delegation. In the experiment we should include as many collaborating machines as possible, and make sure that these machines are, in network layer, far away from each other. However, the execution of such an experiment was impossible due to practical difficulties. Probing and then downloading of a mass amount of URLs from the collaborating machines for the sake of our experiment would take a lot of time, and, even worse, could even be detected as a network attack (i.e. denial of service attack). For this reason, we decided to modify our experiment, as described below. The experiment included the following steps:

1. We constructed an IP address hierarchy (as previously described) and introduced the 12 available machines (spread over the Internet) that would be used for the evaluation (hosting the migrating crawlers).

2. We selected 1000 static URLs randomly from our database, and divided them to two subsets, subset A with 650 URLs and subset B with 350 URLs. Then, we probed all the URLs in both the subsets from the 12 collaborating machines.

3. Following, we used the probing results for subset A to perform, for subset A's URLs, the most optimal delegation of the URLs in the subset (according to the probing results, not according to the download time) to these 12 collaborating machines.

4. Then, we updated the IP addresses hierarchy with these 650 delegations. This update was simple, for each URL we just found the smallest subnet that included the URL, and delegated it to the crawler that from step 3 was identified as the one with the lowest probing time (representing the logical distance) for that URL.

5. Finally, we tried to execute the IPMicra algorithm, using the resulting hierarchy, to delegate the remaining 350 URLs (the URLs of subset B). IPMicra, and our 'trained' hierarchy, did not know, up to now, any probing results for the URLs in subset B. So, we count the requests from IPMicra for probing results.

We also used the probing results for subset B to find the optimal delegation for these URLs, for matters of comparison. Finding the optimal delegation for these URLs was trivial, we just delegated each URL to the crawler with the lowest probing time.

During the experiment, we measured the following parameters: (a) the probing requests for IPMicra during the delegation of the URLs in subset B, (b) the success rate of the URL delegations (compared to the optimal delegations), (c) the difference in the probing time of the URL delegations from the optimal delegation. We repeated the

experiment for 30 times, each time with different sets of URLs, with the same set of crawlers. We also varied the probing threshold in the experiments. Collection of the probing times for the experiment took place in low-use (local) hours of each crawler, in order to avoid creating problems in the local networks (also used from human users). Probing was done with both the suggested metrics, round trip time (with the `ping` tool) and by timing an `HTTP/HEAD` request with very similar results. Due to space limitations, below we present only the results with `ping`.

The experiment had very encouraging results. With a probing threshold set to 50msec, the delegation procedure resulted to 261 optimal delegations (average) from the 350 (75%), with only 1048 probes, while the brute force location aware delegation would require 12*350=4200 probes. Furthermore, the 89 sub-optimal delegations were having an average probing difference from the optimal of less than 13 msec (and a maximum probing time equal to the probing threshold, which was 50 msec), which means that they were very near to the optimal. Moreover, while for practical reasons the experiment did not include many testing URLs (only 350), it was clear from the results that as the hierarchy was getting more 'trained', the average number of the required probes for the delegation of a URL was decreasing. For example, while the average number of probes required for the delegation of all the URLs was 2.99 probes per URL, the average number of probes for delegation of the last 50 URLs (after the hierarchy was 'trained' with the previous 300 URL delegations) was 2.66 probes per URL. For the first 300 URLs the average was 3.05 probes per URL.

These results show that the IPMicra methodology was able to approach the optimal URL assignment, with as less as one fourth of the probes of the brute force approach (described in section 5.3) and still have high quality results. Furthermore, in the cases where IPMicra was making a sub-optimal delegation, the probing results between the delegation of IPMicra and the optimal delegation was negligible (average of 13 msec). Moreover, IPMicra appears very scalable and thus quite suitable for the real Internet for a number of reasons: (a) IPMicra was improving its results and needed less probes since each delegation was also saved in the hierarchy for further use, (b) adding more crawlers in the IP hierarchy is straightforward and results in improving the IPMicra algorithm (more and better distribution of IP subnets), and (c) the system includes dynamic calibration of URL delegations, for discovering and correcting invalid probing results and delegations with negligible overhead.

In order to see how the value of the probing threshold affects our methodology we repeated the experiment with a probing threshold set to 25msec and 100msec, and, as expected, the results differed. In the first case, we required an average of 2051 probes (instead of 4200). The optimal delegations in this case were 313 of 350 (90%), and the average difference of the probing of the suboptimal delegations from the optimal

	Probing=25msec		Probing=50msec	
# of crawlers	optimal	probes%	optimal	probes%
6	558	67	434	45
12	891	54	703	29

Table 8.3: Comparison results with 6 and 12 crawlers. Optimal delegations and percent of required probes compared to the brute force approach with the same number of crawlers (probes/brute-force probes)

ones was 5,6 msec. In the second case, where we set the probing threshold to 100msec, IPMicra required only 617 probes, and we had 187 successful delegations from the 350 (53%). Furthermore, the average difference of the suboptimal delegations from the optimal ones was 29.9 msec. In all the sub-optimal delegations in both the cases, as denoted from the algorithm, the maximum probing time was equal to the probing threshold.

Finally, in order to investigate how the number of the available migrating crawlers affects our methodology, we also ran the IPMicra simulation with a smaller crawlers subset. We ran the experiment with a subset of 6 crawlers, instead of the original 12. While the subset was too small for the real web size, the experiment revealed that the number of web crawlers significantly affects the performance of the IPMicra algorithm. We repeated the experiment multiple times, each time selecting a different subset of 6 crawlers. More precisely, in the experiment with 6 crawlers and probing threshold set to 25msec, on average, we required 67% of the total probes in the original brute-force approach(with 6 crawlers), while, with 12 crawlers, we required only 54% probes of the total probes in the original brute-force approach 8.6. Similarly, when the probing threshold was set to 50msec, on average, with 6 crawlers we required 45% of the probes in the respective case with the brute-force approach, while with 12 crawlers we required only 29% of the probes in the respective case with the brute force approach 8.6. This practically means that the methodology is scalable and can handle an increasing number of migrating crawlers due to the efficient algorithm for selecting the most promising crawler. In fact, we expect that a testing in the real Internet with a much bigger number of crawlers and an experienced hierarchy (with many delegations) will significantly reduce the required probes, since a big portion of our hierarchy will already be assigned to crawlers. Furthermore, in all the tests, the optimal delegations with the 6 crawlers were significantly less than the optimal delegations with the 12 crawlers 8.3.

The above results and our experiments did not include the actual execution (i.e. data crawling) of the suggested delegation for practical reasons. Running such a function in the twelve foreign hosts could be characterized as network attack, and would also

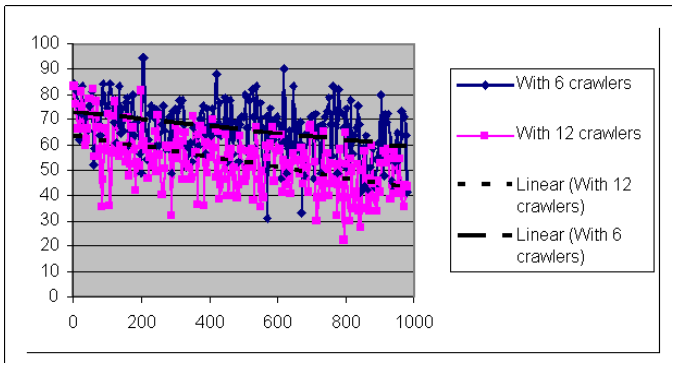

Figure 8.2: Percentage of probes required (compared to the respective brute-force approach) with probing threshold=25msec

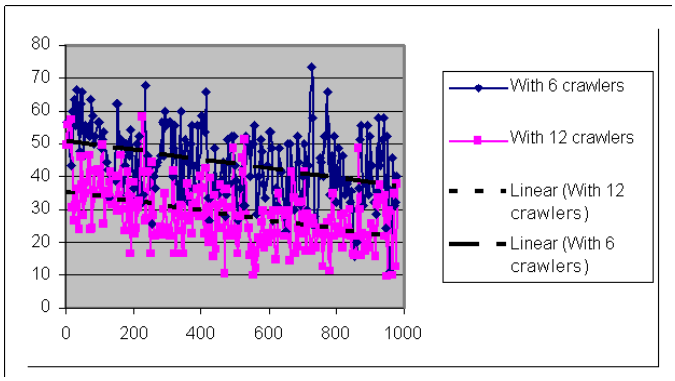

Figure 8.3: Percentage of probes required (compared to the respective brute-force approach) with probing threshold=50msec

take significant time and use important resources, that the collaborating networks could not spare. However, based in the previous experiments (section 8.3 and section 8.2), during which we showed direct correlation of probing and downloading time, we argue that the downloading time for the IPMicra delegation would be very near to the optimal downloading time. Thus, we expect the downloading time to be one order of magnitude less compared to the average downloading time with a random assignment of URLs to crawlers.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

In this work we proposed distributed location aware web crawling, an alternative web crawling method that can importantly alleviate the processing and network bottlenecks which occur in the traditional web crawling systems. The suggested method is an extension of UCYMicra, which was able to distribute the crawling process in a number of collaborative sites, but could not dynamically provide optimal delegation for any newly-discovered URLs.

We proposed and evaluated a number of approaches for distributed location aware web crawling. The evaluation revealed that the brute force approach proposed in this work was able to importantly reduce the time required to download all the web pages (by **one order of magnitude**) (compared to a location unaware distributed web crawler), but required important probing, in order to find the best crawler to take over each domain name. The HiMicra approach, building and using the hierarchy of migrating crawlers to perform the delegations of the domain names to the crawlers, was also able to make sufficient delegations, and at the same time reduce the need for probing importantly. Finally, the IPMicra approach, using information available from the four Regional Internet Registries, was able to provide a very near-to-optimal delegation of URLs to the available migrating crawlers with a very inexpensive manner (importantly less probes). Furthermore, our experiments revealed that IPMicra constantly calibrates and optimizes the hierarchy, thus, improves in quality and reduces the need for probing during time.

We consider this work of significant importance due to the imperative need for a scalable web crawler, that will be able to catch up to the expanding and rapidly changing

web. IPMicra can be the solution to the web indexing problem faced from all the search engines nowadays, this way reducing the time required to download the same set of web pages by **one order of magnitude**, compared to a location unaware distributed web crawler. Furthermore, while the location aware infrastructure developed in this work is powered from UCYMicra, it can be applied (as a framework) to any other (fully or partially) distributed web crawler. The framework can even be applied in existing commercial approaches, like the Google Search Appliance or Grub.

Moreover, the location-aware infrastructure developed in this work can be used as a framework to facilitate several optimizations itself, for generic distributed applications in the Internet. For example, this framework could efficiently enhance the load balancing schemes used from content delivery networks, such as Akamai.

9.2 Future work

While the current work has impressive results, we plan to continue working in the optimization of the suggested approaches. First of all, we will soon improve the difference algorithm implemented in the migrating crawlers, so that only the processed subset of the updated data is transmitted over the Internet, in order to update the central search engine database. Such an improvement can significantly reduce the data transmitted to the search engine's site from the migrating crawlers, and also importantly reduce the workload in the coordinator (for integrating the results back to the database).

Furthermore, we plan to run the experiments in the real Internet with a significantly larger set of migrating crawlers and web pages. Since all the suggested methods are based in the migrating crawlers' high distribution, we expect better results when the number of migrating crawlers is increased. We also expect that both the proposed systems (HiMicra and IPMicra) will be favored from a more uniform distribution of the migrating crawlers. The inclusion of more URLs in our experiment (not only 1000) will also let the IPMicra methodology to calibrate (optimize) for better results, and will help toward a more efficient hierarchy in HiMicra.

Finally, we would like to address some scalability issues in the database component. In our experiments we were always using commercial databases to implement the database component. The commercial solutions could not offer a significant performance for the web crawling task. Thus, in our experiments, since the web crawling techniques alleviated the network and processing bottlenecks, we soon faced a bottleneck in the database component. The web crawling task has specific needs from the databases(batch updates, compression, a big number of concurrent tasks, an enormous file size, distributed database in different file systems). This functionality rarely optimized in generic-use databases. Thus, we soon plan to work on the database of the

system, probably by implementing an existing proposal for web crawlers, in order to be able to test the distributed location aware systems with no database bottlenecks.

Bibliography

[1] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *IEEE Computer*, 19(8):26–34, August 1986.

[2] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. Harvest: A Scalable, Customizable Discovery and Access System. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, 1995.

[3] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The Harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28(1–2):119–125, 1995.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual (web) search engine. In *Seventh International World Wide Web Conference*, 1998.

[5] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1623–1640, 1999.

[6] David Chess, B. Grosof, Colin Harrison, D. Levine, C. Parris, and G. Tsudik. Itinerant agents for mobile computing. *IEEE Personal Communications*, 2(5), October 1995.

[7] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile Agents: Are They a Good Idea? Technical Report RC 19887 (December 21, 1994 - Declassified March 16, 1995), IBM Research Division, TJ Watson Research Center, Yorktown Heights, New York, 1994.

[8] J. Cho and H. Garcia-Molina. Parallel crawlers. In *Proc. of the 11th International World–Wide Web Conference*, 2002.

[9] J. Cho and H. Garcia Molina. The evolution of the web and implications for an incremental crawler. In *26th International Conference on Very Large Database Systems*, 2000.

[10] Junghoo Cho. *Crawling the Web: Discovery and Maintenance of Large-Scale Web Data*. PhD thesis, Stanford University, 2001.

[11] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Trans. Inter. Tech.*, 3(3):256–290, 2003.

[12] World Wide Web Consortium. Http 1.1 protocol, 2003.

[13] P. Evripidou, C. Panayiotou, G. Samaras, and E. Pitoura. The pacman metacomputer: Parallel computing with java mobile agents. *Future Generation Computer Systems*, 18(2):265–280, 10 2002.

[14] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L. Wiener. A large-scale study of the evolution of web pages. In *International World Wide Web Conference*, 2003.

[15] Jan Fiedler and Joachim Hammer. Using the web efficiently: Mobile crawlers. In *Proceedings of the Seventeenth AoM/IAoM International Conference on Computer Science*, pages 324–329, San Diego CA, 1999. Maximilian Press Publishers.

[16] Stefan Funfrocken and Friedemann Mattern. Mobile agents as an architectural concept for internet-based distributed applications - the wasp project approach. In *Kommunikation in Verteilten Systemen (KiVS)*, 1999.

[17] Robert Gray, David Kotz, Saurab Nog, Daniela Rus, and George Cybenko. Mobile agents for mobile computing. In *Second Aizu International Symposium on Parallel Algorithms/Architectures Synthesis*, 1996.

[18] Joachim Hammer and Jan Fiedler. Using mobile crawlers to search the web efficiently. *International Journal of Computer and Information Science*, 1(1):36–58, 2000.

[19] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[20] Jun Hirai, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. Webbase: A repository of web pages. In *The 9th International World Wide Web Conference*, 2000.

[21] Damir Horvat, Dragana Cvetkovic, Veljko Milutinovic, Petar Kocovic, and Vlada Kovacevic. Mobile agents and java mobile agents toolkits. In *Hawaii International Conference on System Sciences*, 2000.

[22] Google Inc. Google, September 2003. http://www.google.com/.

[23] Google Inc. Google search appliance, 2003.

[24] Infospace Inc. Metacrawler search engine.

[25] Objectspace Inc. Voyager 4.0 documentation.

[26] G. Karjoth, N. Asokan, and C. Gulcu. Protecting the computation results of free roaming agents. In *Second International Workshop on Mobile Agents*, pages 223–234, 1998.

[27] David Kotz and Robert S. Gray. Mobile agents and the future of the internet. *ACM Operating Systems Review*, 33(3):7–13, 8 1999.

[28] S. Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400(6740):107–109, July 1999.

[29] LookSmart Ltd. Grub distributed internet crawler, 2003.

[30] P. Lyman, H. Varian, J. Dunn, A. Strygin, and K. Swearingen. How much information?

[31] Paulo Marques, Raul Fonseca, Paulo Simoes, Luis Silva, and Joao G. Silva. A component-based approach for integrating mobile agents into the existing web infrastructure. In *Symposium on Applications and the Internet (SAINT)*, 2002.

[32] Sougata Mukherjea. Wtms: A system for collecting and analyzing topic specific web information. In *International World Wide Web Conference*, 2000.

[33] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[34] Odysseas Papapetrou and George A. Papadopoulos. Aspect Oriented Programming for a component based real life application: A case study. In *Symposium on Applied Computing - Software Engineering track*, 2004.

[35] Odysseas Papapetrou, Stavros Papastavrou, and George Samaras. Distributed indexing of the web using migrating crawlers. In *Proceedings of the Twelfth International World Wide Web Conference (WWW)*, 2003.

[36] Odysseas Papapetrou, Stavros Papastavrou, and George Samaras. Ucymicra: Distributed indexing of the web using migrating crawlers. In *Proceedings 7th East-European Conference on Advanced Databases and Information Systems, Dresden, Germany*, 2003.

[37] Stavros Papastavrou, George Samaras, and Evaggelia Pitoura. Mobile agents for world wide web distributed database access. *Knowledge and Data Engineering*, 12(5):802–820, 2000.

[38] A. Pitsillides, G. Samaras, M. Dikaiakos, and E. Christodoulou. Ditis: Collaborative virtual medical team for home healthcare of cancer patients. In *Information Society and Telematics Applications*, 1999.

[39] Avi Rappaport. Robots & spiders & crawlers: How web and intranet search engines follow links to build indexes.

[40] G. Samaras, M. D. Dikaiakos, C. Spyrou, and A. Liverdos. Mobile Agent Platforms for Web-Databases: A Qualitative and Quantitative Assessment. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 50–64. IEEE-Computer Society, 1999.

[41] George Samaras and Andreas Pitsillides. Client/intercept: a computational model for wireless environments. In *4th International Conference on Telecommunications (ICT'97)*, 1997.

[42] T. Sander and C. F. Tschudin. Towards mobile cryptography. In *IEEE Symposium on Research in Security and Privacy*, 1998.

[43] Altavista search engine. Altavista search engine, 2003.

[44] Altavista search engine. Altavista search engine, basic submit, January 2003.

[45] seti@home. Search for extra terrestrial intelligence, 2003.

[46] Kristie Seymore, Andrew McCallum, Kamal Nigam, and Jason Rennie. Building domain-specific search engines with machine learning techniques. In *AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[47] R. Stata, K. Bharat, and F. Maghoul. The term vector database: Fast access to indexing terms for web pages. In *The 9th International World Wide Web Conference*, 2000.

[48] V. Varadharajan. Security enhanced mobile agents. In *ACM Conference on Computer and Communications Security*, pages 200–209, 2000.

[49] N. Yoshioka, Y. Tahara, A. Ohsuga, and S. Honiden. Security for mobile agents. In *Agent-Oriented Software Engineering*, pages 223–234, 2000.