Development of a Scalable Web Crawler

By Hajime TAKANO* and Nobuya KUBO*

ABSTRACT This paper describes the Nexplorer web-crawler system, which has a scalable architecture. A web crawler is an indispensable component of Web-search services. It should have high performance to gather millions of pages as fast as possible, and be configured to meet various service demands. To meet these requirements, we have designed Nexplorer to be a parallel system and configurable by controlling several parameters. Nexplorer has been used in the practical search service NETPLAZA of NEC, and we have confirmed through experiences there that it has a high enough performance to keep downloaded Web pages as fresh as possible.

KEYWORDS **World Wide Web (WWW), Web crawler, Web search service, HTTP (Hypertext Transfer Protocol), HTML (Hypertext Markup Language), Parallel architecture**

1. INTRODUCTION

The World Wide Web (WWW) has already become the most important software to view worldwide information on the Internet, and it has been used as an infrastructure to build various information services since its invention in 1990. A search service for the WWW is indispensable to find relevant information from a huge number of WWW pages from around the world. Such a service must consist of three functions; gathering WWW pages, storing and managing gathered pages in a database, and searching information by users' queries. A Web crawler is an agent system for gathering WWW pages. Because finding and fetching WWW pages are time-consuming tasks and the number of WWW pages on the Internet exceeds 40 billion, a Web crawler should be designed to gather these pages quickly. In addition, it should have a configurable architecture to build services of various sizes.

To meet these requirements, we have developed a scalable Web crawler, which we call "Nexplorer." Its parallel architecture enables its performance to be improved by adding another workstation according to service demands. This paper describes design issues of Nexplorer, its architecture, and evaluation in practical use in an Internet search service. Section 2 explains the requirements of a Web crawler when building a WWW search service. The architecture of the crawler will be described in Section 3. Usage of the crawler in the practical WWW search service NETPLAZA and an evaluation of the crawler are both described in Section 4, and Section 5 will conclude this paper.

*C&C Media Research Laboratories

2. REQUIREMENTS FOR A WEB CRAWLER

This section explains various requirements for a Web crawler.

2.1 General Requirements

There are reportedly over 430 million hosts on the Internet at the beginning of 1999. This means that over 43 billion pages exist, as each host is assumed to have a hundred pages on average. It is also said that there are over two hundred million pages in JP domain. Because WWW can be thought of as a huge information database, some kinds of search capabilities are required to use it. Actually, we can already use search services, such as Yahoo!, AltaVista, InfoSeek, and Excite.

Because WWW pages are managed only in a Web server all Web pages should be gathered together to find necessary information from all pages in the Web servers. Therefore, a WWW search service consists of the following three functions: 1) gathering WWW pages, 2) storing and managing gathered pages in a database, 3) searching information by users' queries. Obviously, a Web crawler is the system for executing the first function.

The fundamental method of a web crawler is repeating the following steps: finding the Uniform Resource Locators (URL)[2] in a document written in Hypertext Markup Language (HTML)[5,6], going to those URLs, and getting the documents indicated by them. This method is based on the concept that an address, which is linked to a document, is represented as an Anchor Tag ("A" tag) on the HTML. However, the method would be a time-consuming task if it executed sequentially. For example, it can fetch only 864,000 pages a day, even if it took 100 milliseconds for each process, and this means it

ould take several months or more to gather every age on the Internet.

Therefore, the design of a Web crawler for the nternet should reflect consideration on both hard- are configurations using multiple CPUs and a par- llel architecture in software. In addition, the design lso requires considerations on reducing the heavy oad of a Web server, the incomplete Hypertext Transfer Protocol (HTTP)[3,4] response some Web ervers may return, and documents written in in- omplete HTML format.

Furthermore, WWW search services are also necessary even in an enterprise Intranet environment, because many companies establish their own information-sharing tools based on the WWW. For this kind of search service, it is enough that a Web crawler can handle at most a hundred thousand pages. However, the search service is required to refresh its information as fast as possible, because freshness of information is the most important for decision making in an enterprise.

According to the above discussions, requirements for a Web crawler in summary are the following:

- It can find and fetch WWW pages as fast as possible.
- It can follow evolutions of Internet formats such as HTTP and HTML, and should be robust enough to handle incomplete data formats.
- The strategy of crawling should be configurable and controllable.
- It can be configurable to provide performance adequate for the demands of the service.

2.2 Functional Requirements

The basic algorithm to find and fetch WWW pages is the following:

1. Getting the URL to visit from the URL database
2. Accessing the Web server where the URL belongs with an HTTP, and trying to fetch a document that matches that URL
3. Setting the status of access into the URL database as an attribute of the URL
4. Extracting other URLs from a downloaded document, if it is an HTML document
5. Adding extracted URLs into the URL database
6. Repeatedly running from Step 1 to step 5

By executing these steps, the crawler can gather all the pages that are similar to one another on the Internet. In practical use, it should follow some restrictions, such as hostname or domain name, depth from a root directory or start URL, and Content-Type.

To achieve the functions described above, a Web crawler must consist of the following components:

- A URL database storing URLs
- A system to manage the database
- A method to communicate with Web servers
- A method of analyzing HTML documents to extract URLs

Furthermore, there are many requirements for a Web crawler from the viewpoint of service. To meet them, we have considered the following functions:

- Gathering WWW pages as fast as possible
- Giving priorities to important sites
- Giving less priority to unimportant pages
- Filtering WWW pages to download by their content type
- Choosing sites to crawl
- Eliminating a WWW page which includes reserved strings
- Ignoring WWW pages in deeper depth of a directory or link than the specified one

2.3 Performance Requirements

Although the performance of a Web crawler depends on the hardware configuration it works on, we decided to design a Web crawler to get the best performance on a given hardware configuration.

When a 30-second interval is inserted between each access to a Web server to satisfy the Robot Exclusion Standard[7], one connection can access only 2,880 pages a day. Then, if a crawler handles 256 connections at the same time, it can download 737,280 pages a day. Moreover, in consideration of the persistent connection mechanism in HTTP/1.1[4], which is supported by recent popular Web-server software, such as Apache, Netscape Server, and Microsoft IIS, the number of pages accessed in each connection becomes more. Therefore, a crawler will be able to access over a million pages a day, if it has software architecture with parallel connections and hardware sufficient to handle over 256 connections. Of course, the more hardware the system uses, the higher the performance becomes.

3. CONFIGURATION OF Nexplorer

This section describes the configuration of a scalable crawler, Nexplorer.

3.1 Overall architecture

We have designed a Web crawler which we call Nexplorer. It is a parallel system running on several workstations, and is configurable to adapt its performance requirements, by changing the number of workstations, the number of connections, and so on.

As shown in Fig. 1., there are two kinds of machines: an Access Machine (AcM) and a Master Machine (MsM). The following components work together on these kinds of machines.

- Access Database (ADB): a set of database files, each of which is assigned to a server and stores URLs within the server.
- Web Server Management Program (WSMV): assigns a file from the Access Database to a proper AcM.
- Access Database Management Program (ADBM): manages database files as its own ADB.
- Web Server Access Program (WSVA): gets URL from its ADB and repeatedly accesses to the server of the URL.
- Tag Extraction Program (EXTR): extracts HTML Tag attributes from downloaded documents.
- Database Location Management Program (DBLM): sends extracted URLs in EXTR to a proper AcM.
- URL Registration Program (RGST): registers a new URL into ADB.
- System Management Program (SMAN): sets up shared memories and controls other processes.

The processes communicate with each other through a socket or a shared memory as shown in Fig. 1. Table I also explains which processes use a socket and which use a shared memory.

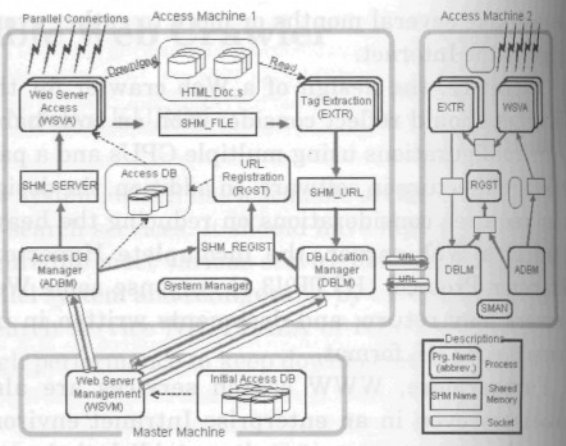

Fig. 1 Nexplorer architecture.

Table I Communication methods.

From	To	Method
WSVA	ADBM	socket
DBLM on a AM	DBLM on others	socket
DBLM	WSVA	socket
EXTR	DDDL	socket
ADBM	WSVA	SHM_SERVER
ADBM	RGST	SHM_REGIST
WSVA	EXTR	SHM_FILE
EXTR	DBLM	SHM_URL
DBLM	RGST	SHM_REGIST

3.2 Role of Each Component

3.2.1 Access Database

The Access Database (ADB) is a URL database in an AcM. It consists of database files, each of which is assigned to only one server. Therefore, URLs on the same server are stored in the database file assigned to the server. Here, a server means a Web server and is represented by the form "<hostname>:<port number>." As mentioned earlier, each AcM has its own ADB and a database file assigned for a server is unique in the crawler system.

To reduce I/O overhead, each database file of the ADB includes a data file and an index file. B+Tree is used for the index here. Because operations to the database file are only inserting a new URL and searching for it, the latest-added URL is at the end of a data file and the position of each record is managed in an index file. This simple construction eliminates any needs for a lock mechanism and thus allows the system to execute as fast as possible.

Note that the initial ADB on the WSVM has a different role. It is a set of URLs used for crawling start points and is distributed to every AcM by the WSVM when the system starts.

3.2.2 Web Server Management Program

Only one WSVM process runs on the MsM. The WSVM assigns each server to one of the AcMs to balance their loads. According to this assignment, the initial ADB and newly found URL are distributed to an AcM corresponding to the server of the URL.

3.2.3 Access Database Management Program

One ADBM process runs on each AcM. The ADBM manages the ADB for an AcM where it runs.

When the system starts, ADBM receives a part of

the initial ADB that the WSVM distributes and saves it as its own ADB. Then, according to a "server list," which is generated from server names assigned to each of the databases in its own ADB, the ADBM fills the shared-memory "SHM_SERVER" with server names gotten from the server list. Because the shared memory works as a ring buffer, the ADBM continuously puts the next server name on the server list whenever there are empty slots in the shared memory until all server names are sent.

Moreover, when the ADBM receives a URL in a server not stored in the ADB, it hands the URL to the RGST through the shared-memory "SHM_REGIST" to make a new database file for the URL.

3.2.4 Web Server Access Program

Plural WSVA processes run at the same time on an AcM. Each process gets a server name from the shared-memory "SHM_SERVER," and opens a database file assigned to the server name. Then, it repeatedly reads the next URL from the database and tries to access the URL to the server.

Because our crawler system follows the "Robot Exclusion Standard," the WSVA checks to see if "/robots.txt" exists on the server just before it accesses the first URL in the database file. If "/robots.txt" exists on the server, the WSVA downloads it, then ignores any URLs that match restrictions described in "/robots.txt."

If the WSVA is going to access a URL it has never visited before, it uses the "HEAD" method of the HTTP request to avoid getting the document itself. By using the "HEAD" method, the WSVA can know if the URL exists or not and which Content-Type (explained below) the URL is. This step is skipped if the URL has already been visited before.

When the URL exists and is worth downloading, the WSVA accesses it by the "GET" method of HTTP to download the document. The document fetched in this step is saved as a local file on the AcM. The local file has a header block in front of the document itself, which stores some fields of HTTP response of the URL, such as Status-Code, Content-Encoding, Content-Type, and Last-Modified. This information will be used to decide whether to access or not in next time, and replaced when the URL is accessed.

The WSVA does not always access to URLs. It decides whether or not to access, according to many condition parameters. Some examples are:

- Depth of directory in URL string
- Inclusion of a special character in URL string
- Status-Code of previous accesses
- Content-Type
- Last-Modified date

3.2.5 Tag Extraction Program

Plural EXTR processes run together on each AcM. The EXTR extracts HTML tag attributes from documents fetched by the WSVA. The WSVA informs the local file names through the shared-memory "SHM_FILE" and the EXTR opens the file and starts extracting.

Although the content of the local file has one of a variety of "Content-Type," the EXTR handles only HTML-formatted documents to find hyperlinks to Web pages. By analyzing the "A" tag and "FRAME" tag of HTML, the EXTR finds URL strings as hyperlinks, and the URL strings are transferred to the DBLM through the shared-memory "SHM_URL."

For some applications, it also extracts several tags, such as "IMG," "APPLETS," "TITLE," "PLUGIN," and "SCRIPT."

3.2.6 Database Location Management Program

One DBLM process runs on an AcM. The DBLM delivers a new URL found in the EXTR to a proper process of its own RGST, DBLMs on other AcMs, and the WSVM on MsM. For this purpose, it makes a copy of the server correspondence list on the WSVM at the start of the system.

In case a server of the URL is included in the ADB on the same AcM, the URL is just sent to the RGST through the shared-memory "SHM_REGIST." If the server is assigned to the ADB on other AcMs, the URL is sent to the DBLM on the AcM through a socket connection. Moreover, if the URL is completely fresh to the system, it is sent to the WSVM through a socket connection.

3.2.7 URL Registration Program

One RGST process runs on an AcM. The RGST reads a URL from a shared-memory "SHM_REGIST," and stores it into a database, if the URL has not been registered yet.

3.2.8 System Management Program

The SMAN manages the start and end of every process of the crawler system. It also keeps and manages shared-memories.

3.3 Further Issues

3.3.1 Robustness against Various Behaviors of Web Servers

There are two popular versions of HTTP, HTTP/1.0 (RFC1945), and HTTP/1.1 (RFC2068).

Because HTTP does not require verifying of its version in the handshake of a communication, a Web server is generally implemented to ignore unknown fields of HTTP. Although the Web crawler speaks HTTP/1.1, many Web servers still understand only HTTP/1.0. Therefore, some such servers return incomplete responses to the requests of HTTP/1.1 from the crawler.

For example, there is a server which returns document data after responding to the "HEAD" method of a request. It is assumed that the server cannot understand the HEAD method of HTTP.

Another example is an action to the URL string of a directory. When the crawler wants to access a directory on a server with a URL string without "/" at the end of the string, some servers return a directory structure. In addition, some others return the redirect response including the correct URL string, which asks users to access again with the correct URL.

To handle this kind of behavior, Nexplorer has been designed to handle incomplete responses from some Web servers.

3.3.2 Robustness against Incompletely Formatted HTML Documents

Almost all HTML documents are still written by hand, and they may have many grammatical errors.

The anchor tag should begin with "<A>" and ends with "</A>." Some common errors, for example, are a sentence beginning with "<A>" does not end with "</A>," or Anchor tag itself is not enclosed with ">."

Moreover, the Japanese encoding system is another problem. There are three popular encoding styles in Japanese documents: ISO-2022-JP, Japanese EUC, and Shift-JIS. Therefore, it is necessary for Japanese documents to handle these three styles, even if a document includes all three encoding styles at the same time.

4. BUILDING A SEARCH SERVICE

Nexplorer has been used in a practical search service NETPLAZA (http://netplaza.biglobe.ne.jp), which is managed by BIGLOBE Personal Sales Division, NEC Corporation. This section describes the service architecture of NETPLAZA and the experimental evaluation done there of Nexplorer.

4.1 Configuration of a Search Service

Like to other search services, NETPLAZA consists of three parts: Nexplorer as a Web crawler, RetrievalExpress[1] as a full-text database, and specialized CGI programs for user interaction. Figure 2 shows an example page of a search result to a query keyword.

Nexplorer crawls WWW pages in JP domain in a predefined period. The documents fetched by each crawling are transferred to RetrievalExpress and it makes full-text indices for quick response to users' queries. Because Nexplorer can currently check millions of WWW pages a day, in spite of a rather small hardware configuration, its performance is fast enough to keep the freshest information for search services.

If the service requires higher performance than Nexplorer provides, it means the service needs to check more pages than it currently does. The performance will be improved by adding another workstation to Nexplorer.

4.2 Experimental Evaluations

Table II shows a performance example of one AcM in Nexplorer. Because the result is from the time

Fig. 2 Example of NETPLAZA search.

Table II Communication methods.

Item	Value	Average per day
Sampling period	17:25 to 19:22 on Mar 18	--
Successful requests	91,718	1,128,824

hen it worked with some restrictions, it does not how peak performance of Nexplorer. However, it is nough to understand that the performance of Nexplorer is potentially almost equal to other Web rawlers used in famous search services, such as AltaVista, HotBot, goo, and InfoNavigator.

5. CONCLUSION

We have developed Nexplorer, which is a Web-crawler system having a scalable architecture. As explained in Section 3, Nexplorer consists of several software components and its configuration is valuable in response to the demand of a search service. Nexplorer has already been used in the service of NETPLAZA, and its crawling performance is fast enough to maintain the quality of the services.

We also plan to apply Nexplorer to small-size Intranet environments to verify its scalable architecture. Moreover, we will extend the functions of Nexplorer to include strategic crawling.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Satoshi Goto and Dr. Shiro Sakata of C&C Media Laboratories, NEC Corporation, for their direction. They would also like to express our gratitude to Dr. Yoshiyuki Koseki and Dr. Tomonari Kanba of C&C Media Laboratories in NEC Corporation for their discussions and supports. Then they would like to thank Mr. Takehiko Shimojima of the BIGLOBE Personal Sales Department of NEC Corporation and other members of the NETPLAZA project for their valuable comments on Nexplorer and for their encouragement to us. They also thank their colleagues in C&C Media Laboratories, NEC Corporation, for their support.

REFERENCES

[1] S. Akamine and T. Fukushima, "Flexible String Inversion Method for High-Speed Full-Text Search," *Proc. Advanced Database Symp. '96*, Tokyo, Dec. 2-4, 1996 (in Japanese).

[2] T. Berners-Lee, et al., "Uniform Resource Locators (URL)," *RFC 1738*, Dec. 1994.

[3] T. Berners-Lee, et al., "Hypertext Transfer Protocol —HTTP/1.0," *RFC 1945*, May 1996.

[4] R. Fielding, et al., "Hypertext Transfer Protocol —HTTP/1.1," *RFC 2068*, Jan. 1997.

[5] T. Berners-Lee and D. Connolly, "Hypertext Markup Language —2.0," *RFC 1866*, Nov. 1995.

[6] D. Raggett, et al., *HTML 4.0 Specification*, *http://www.w3c.org/TR/REC-html40/*, Apr. 1998.

[7] M. Koster, *A Standard for Robot Exclusion*, http://info.webcrawler.com/mak/projects/robots/norobots.html, June 1994.

Received April 6, 1999

* * * * * * * * * * * * * * * * *

Hajime TAKANO received his B.E. and M.E. degrees in electronics and communication engineering from Waseda University in 1988 and 1990, respectively. He joined NEC Corporation in 1990, where he has been engaged in the development of hypermedia systems and window systems, and design of WWW information services. From 1997 to 1998, he was a visiting scholar at the Computer Science Department of Stanford University. He is now Assistant Research Manager of the C&C Media Research Laboratories.

Mr. Takano is a member of the Information Processing Society of Japan.

Nobuya KUBO received his B.E. and M.E. degrees in information and computer sciences from Osaka University in 1989 and 1991, respectively. He joined NEC Corporation and has been engaged in research and development of CSCW systems and WWW information services. He is now Assistant Research Manager of the C&C Media Research Laboratories.

Mr. Kubo is a member of the Information Processing Society of Japan.

* * * * * * * * * * * * * * * * *