Chapter 2

SEARCH ENGINES AND CRAWLERS: A REVIEW

2.1 GENERAL

The Internet \cite{1,2} is a network of networks, joining many government organizations, universities and private computers together with a view to provide an infrastructure for transport of data and messages. Today’s Internet is the result of joint effort of following proponents:

- J.C.R. Licklider of MIT proposed a global network of computers for the purpose of sharing information and resources in 1962.
- Leonard Kleinrock of MIT developed the theory of packet switching, the much needed technology for data transfer across the Internet connections in late 1962.
- Lawrence Roberts of MIT connected a Massachusetts computer with a California computer over dial-up telephone lines in 1965, the first step taken towards development of wide area networks.
- Ted Nelson, an American scientist introduced the terms hypertext in 1965, which allows user to sift between the collections of cross referenced documents.
- Dr. Paul V. Mockapetris of Information Sciences Institute (ISI) invented the Domain Name System (DNS) in 1984, to map complex IP addresses with easy-to-remember names ending in extensions such as .com, .org, .edu, .gov, .mil and country codes including .in for India.
- Tim Berners-Lee of MIT enhanced the work of Ted Nelson and gave the idea of a system of interlinked hypertext documents called World Wide Web (WWW) in 1990.

The above contribution formed the basis of ARPANET at “Defense Advanced Research Projects Agency” (DARPA), a defense initiative of USA towards connection of their computers. The ARPANET was designed to have alternative communication path with a view to provide atleast one available path to access information in the event of a probable
nuclear attack. The starting point for host-to-host communication on the ARPANET was the 1822 protocol [58], which defined how a host computer transmitted messages to an ARPANET router, called an Interface Message Processor (IMP). The 1822 protocol was found inadequate for handling multiple connections among different applications running in a host computer. This problem was addressed with the development of Network Control Program (NCP), the protocol which provided a standard method to establish reliable, flow-controlled, bidirectional communication links among different processes in different host computers [59]. In 1983, TCP/IP protocols replaced NCP as the ARPANET’s principal protocol for communication giving way for the evolution of the present day Internet. In 1990, Tim Berners-Lee gave the idea of an information repository consisting of hypertext documents linked together through hyperlinks. This work was later known as WWW [6]. In fact, WWW is a collection of hypertext documents with cross references which allow the user to sift between them with the help of a browser such as Internet Explorer, Mozilla Firefox etc.

The Internet of the late 1990s, which began with a small military communication network, has evolved into a complex environment, mainly used for five types of operations:

- Long-distance transactions (e.g. e-commerce, form-filling, remote work, entertainment)
- Interpersonal communication
- Data storage
- Research (i.e. data finding) and
- Remote data access and downloading.

The various milestones achieved through evolution of Internet are given in next section.

2.1.1 EVOLUTION OF THE INTERNET

The number of computers connected to the Internet has grown dramatically [3, 4]. Started from the four connected computers at university research labs, today, the Internet connects more than 440 million computers as shown in Fig 2.1.
It may be observed that Internet growth increased exponentially after 1995. It is estimated that there are 1 billion to 1.5 billion Internet users worldwide and the number is continuously growing. The Fig. 2.2 shows the growth of number of users on the Internet since 1995.

The Domain Name System (DNS) is a hierarchical naming system for computers, services, or any resources connected to the Internet. Before the DNS was invented, each computer on the network retrieved a file called HOSTS.TXT from a computer [60]. The
*HOSTS.TXT* file mapped names into numerical addresses. With the rapid growth of Internet, need of a scalable system emerged which could record the change of a host’s address dynamically at one place only. Other hosts would learn about the change through a notification system, thus completing a globally accessible network of all hosts’ names and their associated IP addresses.

In 1998, the U.S. Department of Commerce created Internet Corporation for Assigned Names and Numbers (ICANN) to privatize the operations and registrations of domain names. Since then, different domain name sales have risen nearly 10-fold, but .com is still the most popular domain as shown in Fig. 2.3.

![Fig. 2.3: Change in Domain Name Sale from 2000 to 2008](image)

In 1993, a computer science student Marc Andreessen created the first popular web browser, known as *Mosaic*. Mosaic was first of its kind browser which displays images inline with text instead of displaying images in different window. In fact, mosaic was the first graphical web browser. The development of *mosaic* made the foundation of Netscape Navigator whose descendant is Mozilla Firefox.

At present a number of websites and web pages has been registered to different domains. Fig 2.4 (a) and (b) shows the exponential growth of websites and web pages in last 15 years.
The Internet traffic is growing with a pace of more than 100% a year [18]. Traffic explosion has been noticed first in 1974 when daily traffic on the Internet surpassed 3 million packets. Initially measured in terabytes and petabytes, it is estimated that the future measurement of monthly traffic volumes would be in the exabytes, which is 10 to the 18th power bytes as shown in Fig. 2.5.
It may be observed from the Fig. 2.5 that the monthly Internet traffic which was only 1 terabyte in 1991 has increased to 4000 pentabytes in 2007.

2.2 HYPERTEXT DOCUMENTS

Hypertext Documents [5,6,7], the term introduced by Ted Nelson in 1965, an American computer scientist, is a collection of documents with cross-references which allow the user to sift between them with the help of a browser such as Internet Explorer, Mozilla Firefox and Google Chrome. The prefix hyper to the word text is used to describe the ease and speed at which a user can navigate between related pieces of text. The browser parses the hypertext document, displays the text, and highlights the portions that contain links to other sections and/or documents. To access a hyperlink the user normally clicks on the piece of highlighted text by using mouse or any other pointing device. The browser then navigates to the document pointed to and displays it. On the web, each page has an address known as a Uniform Resource Locator (URL). These URLs can be used to reference a document from any other document.
Hypertext documents are created using a markup language [7, 8]. A markup language gives structure to a document by embedding components such as hyperlinks, titles and headings etc at suitable places in the document. For instance, in Hyper Text Markup Language (HTML) tags are embedded in the text to give the document structure and formatting, thereby allowing data to be presented in a non-linear fashion that differs greatly from printed media. In fact, hypertext was one of the biggest developments that became the basis for the tremendous popularity of the today’s Internet.

2.2.1 THE STRUCTURE OF HTML DOCUMENT

Each HTML document is divided into two main parts a Heading and a Body [8, 9] as shown in Fig. 2.6. The arrows indicate that the source element may also contain the target element.

The heading contains information to identify the page such as header, while the body contains the actual information to be displayed. Like other languages used with computer, HTML [8, 9] has its own rules of grammar. To separate HTML instructions from text to be displayed, each instruction consists of characters surrounded by less than (<) and
greater than (>) symbols. The resultant group of characters is known as a Tag. For example, the HTML tag <BR> instructs a browser to begin a new line. An author uses tags to tell the browser which part of the page corresponds to the heading and which part corresponds to the body. The less than (<) and greater than (>) characters are used to bracket the name of the tag in pairs. One of the pair is used to start the section and another to close it. Both tags in a pair have the same name except the closing tag that includes a slash (/) before the tag name. For example, the pair of tags: <HEAD> and </HEAD> bracket the heading. Similarly, the pair of tags: <BODY> and </BODY> bracket the body. Another example of paired tag occurs inside a heading, where the pair of tags <TITLE> and </TITLE> bracket the document title.

Fig. 2.7 illustrates the arrangement of paired tag in a typical web page. Of course, browser does not display the tags for a user to see. Instead, the tags merely control the browser output.

```
<HTML>
  <HEAD>
    <TITLE>
      Format document title goes here ...
    </TITLE>
    <TITLE>
      Additional heading information, if needed
    </TITLE>
  </HEAD>
  <BODY>
    Actual information to be displayed on the page
  </BODY>
</HTML>
```

Fig. 2.7: The Basic Tags used to Divide a Web Page into a Heading.

It may be noted here that in Fig. 2.7, each tag has been placed on a separate line and the information in both the heading and the border are indented. Furthermore, pairs of tags such as <HEAD> and </HEAD> are indented exactly the same amount. Though HTML uses free format input, the indentation has been used to enhance readability.
2.2.2 THE LINKS IN A HTML DOCUMENT

In order to make it possible for an individual to record the location of a document, the World Wide Web assigns each page of information a unique identifier called Uniform Resource Locator (URL), which consists of a short character string that identifies a particular multimedia document. When a browser displays a page of information, it also displays the URL for that page. Given a valid URL, a browser can go directly to the page without passing through other documents. For example, the following URL

www.hotmail.com

refers to the home page of hotmail, the popular free email server. A frequently accessed page can be accessed via a link called bookmark. Each URL contains the following information:

- Name of the protocol to use while accessing the server.
- Domain name of the remote computer on which the server is running.
- Name of the item to be requested from the server.

Fig. 2.8 illustrates how the URL encodes the information.

![URL Format Diagram]

The first part of the URL specifies an access protocol that tells the browser how to contact the remote server. The domain name of the computer on which the server runs follows the colon and two slashes. Finally a slash is used to separate the computer name from the suffix that identifies a specific item. A detailed discussion on domains of sites is given in following paragraph.

The domain name system [14, 16] uses a hierarchical naming scheme known as domain names. The root of the DNS tree is a special node with a null label. The domain name of any node in the tree is the list of labels, starting at that node, working up to the root, using
a period ("dot") to separate the labels. For example, the domain name "ymcaie.ac.in" contains three labels: "ymcaie", "ac", and "in". In this example, the lowest level domain is "ymcaie.ac.in" (the domain name for the YMCA University Academic organization in India), the second level domain is "ac.in" (the domain name for Academic organizations of India), and the top level domain is "in" (the domain name for India). Fig. 2.9 shows hierarchical organization of DNS.

Every node in the tree must have a unique domain name, but the same label can be used at different points in the tree. The top-level domains are divided into three areas:

- arpa is a special domain used for address-to-name mapping.
- The seven 3-character domain names (generic (organizational) domains).
- The 2-character domains are based on the country codes. These are called the country (the geographical) domains.

The seven 3-character generic domains are listed in the Table 2.1.
Table 2.1: Three-Character Generic Domain

<table>
<thead>
<tr>
<th>Domain Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM</td>
<td>Commercial organizations</td>
</tr>
<tr>
<td>EDU</td>
<td>Educational institutions</td>
</tr>
<tr>
<td>GOV</td>
<td>Government institutions</td>
</tr>
<tr>
<td>MIL</td>
<td>Military groups</td>
</tr>
<tr>
<td>NET</td>
<td>Major network support centers</td>
</tr>
<tr>
<td>ORG</td>
<td>Organizations other than those</td>
</tr>
<tr>
<td></td>
<td>above</td>
</tr>
<tr>
<td>INT</td>
<td>International organizations</td>
</tr>
</tbody>
</table>

In order to refer to the related documents, hyperlinks (i.e. URLs) are embedded in the hypertext documents at suitable places. HTML provides anchor tags `<a>` and `</a>` for the inclusion of hyperlinks. To specify the page to which a given link points to, the initial tag contains the keyword `href`, followed by an equal sign (=) and a URL enclosed in double quotes. The general format of the anchor tag is `<a href = "value"> </a>.

The user can specify links to a page within a document, or the same site but to a different document, or on a different site altogether. Accordingly, the hyperlinks (see Fig. 2.10) can be of two types as defined below [10]:

**Internal link** : The Internal link can be of two types: Page link and Local link.
- **Page links** : The links to the same page of the web document are called page links.
  Example: `<a href="#top"> </a>`
- **Local links** : The links to the pages on same web site are called as local links.
  Example: `<a href="default.htm"> </a>`

**External links** : The links that refer to other websites are called the external links.
Example: `<a href="http://www.example.com/default.htm"> </a>`
2.3 THE WORLD WIDE WEB (WWW)

The World Wide Web [10, 11, 12] (WWW) is an interlinked collection of documents formatted using Hyper Text Markup Language (HTML). These documents contain hyperlinks to other documents. The links can point to a document on the same machine or to one on the other machine residing on the Internet. When a user accesses a web page using its URL, the documents are transferred to the client machine using Hyper Text Transfer Protocol (HTTP). The browser interprets the document and makes it available to the user. The user may follow the links in the presented page to access other pages. The initial World Wide Web program was developed in November of 1990 with the following objectives [6]:

- The provision of a simple protocol for requesting human readable information stored in remote systems accessible using networks.
- To provide a protocol by which information could automatically be exchanged in a format common to the information supplier and the information consumer.
- The provision and maintenance of collections of documents, into which users could place documents of their own.
- To allow documents or collections of documents managed by individuals to be linked by hyperlinks to other documents or collections of documents.
• The provision of a search option, to allow information to be automatically searched for by keywords, in addition to being navigated to by the following of hyperlinks.

• To use public domain software wherever possible and to interface existing proprietary systems.

From a practical viewpoint the WWW is a combination of three technologies: a means of providing useful information such that it can be accessed by the users who are distributed or distinct; a means for users to access information stored at distributed sites without requiring knowledge of the underlying access mechanism; and a means for structuring information such that it can be discovered, retrieved and viewed when desired. These three technologies are enabled by WWW server and client programs. The Hypertext Transfer Protocol (HTTP) is used by distributed servers to communicate with each other and a variety of client applications. Many of these clients are also capable of communicating with each other through information services such as FTP, Gopher and Text search system e.g. Wide Area Information Server (WAIS) as shown in Fig. 2.11. A client can send commands to any server accessible via TCP/IP connection. The command is usually a request for transfer of an information object, which is then displayed locally by the client.

![Client Server Architecture of WWW](image-url)

**Fig. 2.11: Client Server Architecture of WWW**
An information object is identified by a Uniform Resource Locator (URL), which in its canonical form can include the access scheme, an IP/host name address and TCP part for the server location, and a name recognizable by the server representing that object. In most cases, the URL is embedded in other documents as a hypertext reference (link) associated with some meaningful text pointer (anchor). Viewed graphically, these links form a hypertext “web” of related information. From earlier sections, it can be noted that HTML is used to structure information such that it can be readily displayed by viewing clients. Because these clients exist on heterogeneous platforms and may vary in their rendering abilities, HTML emphasizes the description of content and structure rather than form. It defines portions of a document as being hypertext i.e. pieces of text or images that are linked to other documents.

2.3.1 WEB BROWSER

A web browser provides an interface to a user through which the user can find, retrieve, view, and send information over the Internet. A typical browser provides the following basic operations:

- Send and receive worldwide electronic-mail (or e-mail) messages.
- Browse the World Wide Web (or Web), a huge source of publicly indexable information.

Additionally, the current browsers also provide the latest features such as tabbed browsing, spell checking, pop-up blocking, incremental find, live bookmarking, a download manager, private browsing and security. A tab is a navigational control for switching between different sets of web pages. By using tabbed browsing facility a user can have multiple web pages opened in a single window. In incremental find as the user types text, one or more possible matches for the text are found in the current web page and immediately presented to the user. Live bookmarking allow users to dynamically monitor changes to their favorite news sources. Private browsing feature lets users browse the web without leaving any traces in the browsing history. Security feature
prevents the phishing attacks and provides an option of clearing the private data such as browsing history and cookies.

Example: Microsoft Internet Explorer, Mozilla Firefox and Google Chrome.

2.4 SEARCH ENGINE

An end user who wants to use WWW, the vast source of information, must learn to find the information through thousands of servers that are available on line. Normally, the user must specify an initial starting point i.e. an initial address of the website using which the browser takes the user to the target website containing the required document. From here onwards, at each step, the user must choose a link available on the page of the document to move to another part of the document or to some other document being referred on the page. This process of finding the required information is called as browsing or navigation. However, the size of the Internet makes it impossible to find the information by searching one site at a time.

Automated searching differs from browsing in the sense that it does not require a human being to specify a set of remote computers for navigation of the net. Instead, automated search tools use computer programs to find web pages that contain information related to a given topic. The job of the user becomes easy when automated search tools find information on remote computer without requiring the user to look through web pages. Such tools provide a list of the sites containing the information related to the specific topic and the user browses the same afterwards.

In fact, search tools allow one to automatically locate [16, 17]:

- Web pages that contain information about a particular topic.
- Web pages associated with a particular company or individual.
- Web pages that contain information about a particular product or brand.

To make retrieval convenient and quick, the results of a search are returned in the form of a web page that has links to each of the items that was found.
The automated searching needs a computer program that can automatically contact other computers on the Internet, search for specified information, and report the results. Such a program is called a search tool, indexing tool, or a search engine [16]. When the user enters a filename, the search engine apparently searches all computers on the Internet and find files with names that match the specified name. This kind of searching through Internet takes many hours, much larger than one is willing to wait. But the search engine returns with the results after a few seconds only. This is because the search engines gathers information before it is needed with the help of a program called crawler which contacts computers on the internet, collects a list of available information, sorts the list and then stores the result on a local disk on the computer that runs the search server. When the user invokes a search, the server does not need to contact all the computers on the network but consults the list of file names on its local disk similar to the looking up of the answer in a telephone book.

2.4.1 SEARCH ENGINE SYSTEM ARCHITECTURE

The architecture [16] of a common Web search engine contains a front-end process and a back-end process, as shown in Fig. 2.12.

In the front-end process, the user enters the search words into the search engine interface, which is usually a web page with an input text box. The application then parses the search request into a form that the search engine can understand, and then the search engine executes the search operation on the index files. After ranking, the search engine interface returns the search results to the user. In the back-end process, a spider, robot or crawler fetches the web pages from the Internet, and then the indexing subsystem parses the web pages and stores them into the index files.

A general web search engine as shown in Fig 2.12 has three parts i.e. Crawler, Indexer and Query engine. The web crawler (also called robot, spider, worm, walker or wanderer) is a module that searches the web pages from the web world. These are small programs that pursue the web on the search engine's behalf, and follow links to reach different
pages. Starting with a set of seed URLs, crawlers extract URLs appearing in the retrieved pages, and store pages in a repository database.

The indexer extracts all the uncommon words from each page and records the URL where each word has occurred. The result is stored in a large table containing URLs, pointing to pages in the repository where a given word occurs. Indexing methods used in web database creation are: full text indexing, keyword indexing and human indexing. In full text indexing every word on the page is put into the database for searching. It provides a vast result in the response of a query and helps user to find every example for a specific name or terminology. But for a general topic search has to refine the results at their own from lots of false responses. In keyword indexing only important words or phrases are put into a database. In human indexing a person examines the page and determines a very few key phrases that describes it. This allows the user to find a good start of works on a topic, assuming that the topic was picked by the human as something that describes the page.

Fig. 2.12: Web Search Engine Architecture
The query engine is responsible for receiving and filling search requests from users. It relies on the indexes and on the repository. Because of the web's size, and the fact that users typically only enter one or two keywords, result sets are usually very large.

2.4.2 TYPES OF SEARCH ENGINES

There are mainly three types of search engines [17, 98, 99] namely Crawler based search engines, Human powered directories and Meta search engines. A brief discussion on each is given in the following sections:

2.4.2.1 Crawler Based Search Engines

A crawler based search engine creates and maintains its collection at a local repository. At the back end, crawlers continuously download, index and store the documents in repository. Such search engines widely cover the web and often retrieve a lot of information. In fact, a search with in the collection of previously loaded document is made with a view to rank them before being presented in response to a user query. Such search engines contain full/partial text of web pages they linked to. Examples: google, altavista, lycos. etc.

2.4.2.2 Human Powered Directories

Unlike a crawler based search engine, directory based search engines use human editors to create its repository. In fact the rank and placement of the page in repository is not dependent on the placement of keywords but dependent on the editor who decides whether the contents of the page are appealing and valueble to the user/ searchers. The editor organizes the pages in subject categories based on classification of subject. Some directory-based engines charge a fee for a site to be reviewed for listing and indexing. Such directories do not contain full text of the web page they link to, and the coverage is limited as compared to crawler based search engines. In the early 2000s, more leading search engines were relying on human editors in combination with findings obtained with crawlers. In present scenerio search engine like LookSmart, AltaVista, MSN, Excite and yahoo rely on providers of directory data to make their search results more meaningful.
2.4.2.3 Meta Search Engines

The working of a Meta search engine is based on the fact that the web is too large for any one search engine to index it all. So, unlike crawler based search engines, Meta search engine do not crawl the web themselves to build listings instead Metasearch engines create what is known as a virtual database. They send user requests to several other search engines and/or databases for accumulating and aggregating the results into a single list and provide the results to the user. Examples: highway61, cnet, search, metacrawler etc.

2.4.3 SEARCH ENGINE DESIGN ISSUES

A search engine primarily performs three basic tasks:

- Visits the servers on the web to download documents, a task performed by the crawler.
- Generate an index based on embedded keywords in the downloaded documents.
- In response to user query, search the query words in the index to provide links to the corresponding sites containing the documents.

The www is a vast resource of information containing huge amount of dynamic data in the form of HTML pages. Moreover, there is a tendency that users often only look at the results set that can be seen without scrolling, and the results which are not on first page are nearly invisible for the general user. So a search engine should not only download the document at fast rate but also ensure that quality pages are stored and maintained in its repository.

Factors that determine the quality of a search engine are freshness of contents, index quality, search features, retrieval system and user behavior.

Other characteristics [16, 100] that a large search engine is expected to have are scalability, high performance, politeness, continuity, extensibility and portability. Scalability refers to the characteristics by which a search engine can match up with the
exponential growth of the web. A search engine should be continuous in its working. *Extensibility* ensures that the other functionality can be added in to the existing search engine time to time and *Portability* make a search engine platform independent.

In addition to this, search engines should also be able to index documents written in multiple formats, as each file format provides certain difficulties for the search engines. Following are the challenges for a search engine towards the efficient management of its repository:

- **Distributed data:** Data is distributed across different sites and platforms consisting of heterogeneous networks differing on account of different topologies and protocols.

- **High percentage of volatile data:** A large number of documents on the web are dynamic in nature i.e. their contents change at a regular intervals. Thus the search engines need to regularly update its repository with the fresh information.

- **Large volume:** The growth of data is exponential causing scaling issues that are difficult for the search engine to cope up.

- **Unstructured and redundant data:** All documents on the web are not hypertext documents. The job of search engine becomes over challenging in a situation where the documents are also unstructured.

- **Quality of data:** A lot of web pages do not pass through any editorial process rendering the web pages to be false, inaccurate, outdated, or poorly written.

In the light of the above discussion, it may be noted that downloading of documents from widely spread web sites containing volatile information is a challenging task especially when the web consisting of heterogeneous sites and platform. Since crawler is the component of search engine responsible for downloading of documents, its design assumes utmost importance. A detailed discussion on crawlers is given in the next section.
2.5 CRAWLERS

A crawler is a program that downloads pages and stores them in repository maintained at search engine side. Crawler is supplied with a set of URLs for the purpose of downloading the corresponding documents. From this queue, the crawler removes a URL, downloads the page, extracts the URLs embedded therein, and stores them on a buffer at the search engine side and so on. Collected pages are further indexed by an indexer.

Fig. 2.13 shows the flow chart for the working of a typical crawler.
The algorithm of the typical crawler is given below:

Step 1: Remove URL from the queue of seed URLs.
2: Resolve the IP address for the target server using DNS.
3: Download the Robot.txt* file from the server and check for permission.
4: Determine the protocol of underlying host like http, ftp, gopher etc.
5: Based on the protocol of the host, download the document.
6: Identify the document format like doc, html, or pdf etc.
7: Extract the links or references to the other web sites from that documents.
8: Store the document and the URLs in search engine buffer
9: Repeat step 1 to 8 till the queue is empty.

* Robot.txt file carries the downloading permissions and also specifies the files to be excluded by the crawler

Starting with the seed URL set, the crawler follows all links found in that HTML page. This usually leads to more links, which are supposed to be followed again, and so on. A site can be seen as a tree-structure, the root is the start-URL; all links in that root-HTML-page are direct sons of the root. Subsequent links are then sons of the previous sons (See Fig 2.14).

Fig. 2.14: A Tree Structure of a Website
It may be noted that a traditional web crawler doesn’t actually move around to different servers/sites available on the web. It simply sends HTTP requests for the target documents housed on various site.

The crawler retrieves pages from the Web and thereafter the indexer analyses the pages for the purpose of efficient storage at search engine side. Based on their page retrieval and documents refreshing techniques crawlers can be divided into several categories, some of them are discussed here in next section.

2.5.1 TYPES OF CRAWLERS

A web crawler has to deal with two main issues:

- a good crawling strategy for deciding which pages to download next and
- to have a highly optimized system architecture that can download a large number of pages per second.

On other side, it has to be manageable, considerate of “resources and web servers” and robust enough against possible crashes. Several web crawling techniques are in use that differs in their mechanism, implementation and objective. Some of the common web crawling techniques are described in following section.

2.5.1.1 Focused Crawler

A focused crawler [21, 22] attempts to download only those web pages which are relevant to pre-defined topic(s). The goal is to select links that lead to documents of interest, while avoiding links that lead to irrelevant topics. Unlike an exhaustive crawler which follows every link on a page in breath-first manner, the focussed crawler gives priority to links that belong to pages classified as relevant. In fact, it uses an additional classifier to select the most promising links on a relevant page and the crawling cycle starts with a seed list which contains URLs that are relevant to the topic of interest. The documents relating to URLs are downloaded and the links embedded therein are evaluated. The most promising links based on both the content of the source pages and
the link structure of the web is followed and so on. Three main components of a focused crawler are Classifier, Distiller and download workers as shown in Fig. 2.15.

Classifier makes relevance judgments on pages crawled and takes decision whether to expand the link or not. Distiller determines a measure of centrality of crawled pages to determine visit priorities. Download workers with dynamically reconfigurable priority control is governed by the classifier and distiller. The basic idea is to classify crawled pages with categories in topic taxonomy. The relevance rating system uses a hypertext classifier to update the metadata with topic information from a large taxonomy. A user marks interesting pages as they browse, which are then placed in a category in the taxonomy.

A form focused crawler [21, 24, 25] shown in Fig. 2.16 deals with the sparse distribution of forms on the web. It avoids crawling through unproductive paths by limiting the search
to a particular topic, learning features of links and paths that lead to pages that contain searchable forms, and by employing appropriate stopping criteria.

![Diagram of a Form Focused Crawler]

Fig. 2.16: General Architecture of a Form Focused Crawler

The crawler uses two classifiers to guide its search, the page classifier and the link classifier. A third classifier, the form classifier, is used to filter out useless forms. The page classifier is intelligent enough to classify whether a page belonging to the topic in taxonomy or not. The link classifier identifies the links that are likely to lead to pages that contain searchable forms. Forms that are searchable and not already present into the database are selected by the form classifier and added to the form database.

### 2.5.1.2 Incremental Crawlers

In order to refresh its collection, a traditional crawler periodically replaces the old documents with the newly downloaded documents. On the contrary, based upon the estimate as to how often pages change, an incremental crawler downloads only the changed pages and replace the old one in the existing collection of search engine database thereby maintaining the freshness of the information [20, 89, 92]. A discussion on the prevalent incremental crawlers is given below:
a) Incremental Crawler

The architecture of incremental crawler proposed by Junghoo Cho et. al. is shown in Figure 2.17.

Architecture is composed of the following Modules/Data Structures:

- **All_Urls**: A set of all URLs known
- **Coll_Urls**: A set of URLs in the local collection.
  (it is assumed that this list is full from the beginning)
- **Collection**: Documents corresponding to the URLs of Coll_Urls list.
- **Ranking Module**: It constantly scans through the All_Urls and Coll_Urls to make the refinement decisions.
- **Update Module**: It takes a URL from the top of the Coll_Urls and decides whether the document corresponding to that URL has been refreshed or not.
- **Crawl Module**: Add URLs to All_Urls and updates the Collection.

Based on some previous history or estimate, the ranking module picks a URL from All_Urls list and assigns rank to it. The ranked URLs are placed on the list called
Coll_Urls. The update module picks the URLs from Coll_Urls list in the order of their rank and crawls/updates the corresponding document. Links contained in the fresh downloaded documents are added in to the All_Urls list and so on.

b) A Novel Approach towards Efficient Management of Volatile Information

When the information contained in a document changes very frequently, the crawler downloads the document as often as possible and updates it into its database so that fresh information could be maintained for the potential users. If the number of such documents is large, the crawling process becomes hopelessly slow and inefficient because it puts a tremendous pressure on the internet traffic.

In order to reduce the traffic on the web and increase the efficiency of the crawler a novel approach for managing the volatile information was introduced [18]. The technique introduces volatile information tags in HTML documents to store the changing information also called as volatile information. While storing the document at server side the Vol# tags are extracted from the document along with their associated volatile information. The tags and the information are stored separately in a file having same name but with different extension (.TVI) [10, 18]. The TVI (Table of Variable Information) file is updated every time the changes are made to the hypertext document. This file containing the changed contents of a document is substantially smaller in size as compared to the whole document.

So, for maintaining fresh information at the search engine side, the crawler needs to bring only the .TVI file which would definitely ensure minimization of network traffic. For example, consider a web page of a property dealer's website containing volatile information about the rates of the property designed through XML as shown in Fig. 2.18. Assume this page changes every week.
Fig. 2.18: An XML Document containing Volatile Information

All the volatile information for this web page has been marked with Vol. tags and stored in the separate TVI file which is small in size as compared to the full document as shown in Table 2.2.

Table 2.2: TVI File containing Volatile Information

<table>
<thead>
<tr>
<th>Vol. Tag #</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vol1</td>
<td>12/02/2010</td>
</tr>
<tr>
<td>Vol2</td>
<td>Friday</td>
</tr>
<tr>
<td>Vol3</td>
<td>10 K/SqM</td>
</tr>
<tr>
<td>Vol4</td>
<td>50 K/SqM</td>
</tr>
<tr>
<td>Vol5</td>
<td>1 K/SqFt</td>
</tr>
<tr>
<td>Vol6</td>
<td>3 K/SqFt</td>
</tr>
</tbody>
</table>

It has been found that the size of a TVI file is on an average 5% of the size of its corresponding main hypertext document.

The hypertext documents that support TVI and other related augmentations are called as Augmented Hypertext documents.
2.5.1.3 Hidden Web Crawler

A large amount of information on the web today is available only through search interfaces (forms) [64, 65, 92]. Hence a huge amount of data remains invisible to the users. For example, if a user wants to search information about a flight for a particular destination, then he/she must go to airline site and fill the details in a search form. As a result he/she would get the details of the available flights. This information is referred to as hidden and the related part of the web which was earlier hidden behind the search interfaces is called hidden web. A typical search engine cannot discover and index such pages as they are dynamically generated on the request of the user through the search interfaces.

The above discussion indicates that identification of search interfaces is an important activity. As for as development of information retrieval tool for hidden web are concerned, a deep web crawler follows the following steps to download information in the repository.

**Step 1:** Parses the web pages to identify the forms.

**Step 2:** Parse and process the form to build an internal representation.

**Step 3:** Generate the best values to be assigned to the various form elements and submit the completed form using the assignment.

**Step 3** Wait for response pages.

**Step 4** Analyze the response page. Report errors if any, else use the received information as feedback for next iteration. If the response page contains hypertext links, follow them immediately and recursively, to a pre-specified depth.

It may be noted that the links in the response page are also added to the URL queue. However, for ease of implementation, the response pages are also navigated immediately and that too, only up to a depth of 1. Repeat Step 2 and Step 3 until desired results are obtained.
Architecture of the deep web crawler is shown in Fig. 2.19. It includes six basic functional modules and two internal data structures. To start with, the URL list is initialized to a seed set of URLs. The crawl manager decides as to which link to visit next, and makes the network connection to retrieve the page from the Web. It sends the downloaded pages to the Parser module that in turn extracts hypertext links from the page and adds them to the URL List. Any search interface form found thereof is handed over to form analyzer. This sequence of operations is repeated until some termination condition is satisfied.

In fact, to process forms and extract hidden web contents, deep web crawler employs following additional modules and the LVS table.

1. *Form Analyzer* is used to parse and build an internal representation of the form.
2. *Form Processor* assigns appropriate value to various form elements and submit the form to get the response pages.

3. *Response Analyzer* analyzes the response pages to check the validity of the result pages.

4. *LVS (Label-Value Set) Manager* is responsible for insertion and extraction of values from the LVS table. It provides an interface for various application-specific data sources to supply new entries to the table.

### 2.5.1.4 Agent Based Crawlers

As discussed in chapter 1, mobile agents are software processes capable of roaming on wide area networks such as WWW, interacting with foreign hosts, gathering information on behalf of its owner and coming back after having performed the duties set by its user.

The key characteristics of mobile agents are as follows:

- **Migration**: Mobility is the characteristic that allows agents to move between network nodes, but migration is the function which controls how this transfer is achieved. A Mobile agent is different from a normal executing process in the sense that not all of its instructions have to be executed on the same node. Unlike agent migration differs from process migration, the difference lays in the designation of the objects which decides where and when to move. In process migration, migration is normally forced upon a process by the system, due to resource location, load balancing and other similar factors. With mobile agents, it is the agent who decides to move and the underlying infrastructure must support and execute this request. The agents can be moved between the nodes as *state-oriented* or *stateless*. State oriented agents can move to other destination at any point of their execution. Before migration they simply preserve their state and after reaching to the destination they resume execution exactly from the point where they stopped before the migration. Stateless agents start their execution from the beginning of their code rather than the point of migration.

- **Data Acquisition**: A Mobile agent collects the necessary information from its current environment. This information needs to be filtered locally by the agent
before it is either stored with the agent or forwarded to some receiving destination (such as the original network node of the agent). Therefore, mobile agent systems need to have an approval from the system on which they are working.

- **Route Determination:** Once an agent has finished with a network node, it must make a decision as to where to move next. This decision can be taken based on any of the following applicable scenario:

  i) *Predetermination:* In this scenario the agent is given the destinations that it must visit when it is launched.

  ii) *Dynamic Determination:* In the dynamic determination scenario agent has been given complete freedom over the network nodes that it can visit. The mobile agents may make a decision based on its own node knowledge history or information gained from other agents.

  iii) *Hybrid Determination:* This is a combination of pre and dynamic determination methods. The mobile agent is given a list of destination nodes that it must (or must not) visit, but can still exercise some judgment if a potential node offers data or services that are compatible with the agent’s goals. Alternatively, the agent is equipped with a set of criteria that determines the type of nodes that it can visit. Nodes that do not fit these criteria cannot be visited by the agents. This is useful in situations where the speed (and may be quality) of data returned is more important than the depth of data covered.

- **Communication:** The ability for agents to communicate is fundamental to mobile systems. There are following two methods for agent communication to take place:

  i) *Network- oriented Agents:* In this case communicating agents need not reside on the same node or on the same network. Agents communicate through some network-based mechanism, such as message passing.

  ii) *Node- oriented Agents:* These agents communicate through some local inter-process communication mechanism, such as files, shared memory or
anonymous pipes i.e. the communicating parties must be currently executing on the same network node.

Some major advantages of mobile agents are discussed below:

- **Bandwidth:** Distributed systems often rely on communications protocols that involve multiple interactions to accomplish a given task. The result is a lot of network traffic. Mobile agents allow packaging of a process and dispatching it to a destination host where the interactions can take place locally, as shown in Fig. 2.20. Mobile agents are also useful when it comes to reducing the flow of raw data in the network. When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data rather than transferred over the network. So an agent moves the computations to the data rather than the data to the computations.

![Diagram showing the difference between RPC Based Approach and Mobile agent Based Approach](image)

**Fig. 2.20: Mobile Agents reduce Network Load**

- **Latency:** By migrating to the location of the resource, a mobile agent can interact with the resource much faster than from across the network.

- **Asynchronous task execution:** While the agent acts on behalf of the client on a remote site, the client may perform other tasks. Tasks can be embedded into mobile agents, which can then be dispatched into the network. After being
dispatched, the mobile agents become independent of the creating process and can operate asynchronously and autonomously, see Fig. 2.21. The mobile device can reconnect at some later time to collect the agent.

![Diagram showing mobile agents](image)

**Fig. 2.21: Mobile Agents allow Disconnected Operation**

- **Fault Tolerance:** In the event of a network or server failure in a client/server model, it is difficult for the client to reclaim the situation and re-synchronize with the server because the network connection will have been lost. The mobile agents are ideally suited for mobile computing in which computers can be disconnected from the network for long periods of time as they do not maintain permanent connections and their state is self-contained. Instead of being online for a longer period, a mobile user may develop an off-line agent request and launch it as soon as the connection is established. Moreover, it may receive back the agent with the result at some later time.

- **Peer-to-peer communication:** One of the major limitations of the client/server paradigm is that servers do not communicate directly with other servers. Whereas the agent can play both the roles i.e. the client as well as server. For example, when a mobile agent is collecting information from a resource it acts...
as client. However, when another mobile agent wishes to query it, then it becomes a server. This allows for great flexibility in dealing with network entities and distributed resources.

The following section details the most prevalent systems currently available.

a) Mobile Agent Systems

1. Agent TCL

The architecture of Agent TCL (Tool Command Language) [34] is based upon the server model consists of four levels, as illustrated in Fig. 2.22.

<table>
<thead>
<tr>
<th>Mobile Agents</th>
<th>Interpreter</th>
<th>Interpreter</th>
<th>Interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP/IP</td>
<td>E-Mail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 2.22: The Agent TCL Architecture

A brief description of major components is given below:

**Mobile Agents:** - All the services that are available within the system are provided by agents, transportable or stationary.

**Interpreter:** - The execution of agents is handled by an interpreter that is appropriate to the source language of the mobile agent. In Agent TCL, the interpreter of TCL was extended to support three extra modules Security, State and Server Application Programming Interface (API) Module. The Security module prevents an agent from performing malicious actions; State capture module packages and restores the internal state of an agent and Server API module allows interaction with the server (next level) to provide migration and agent communication.
Server: - It handles the management of local agents and incoming agents. The server also provides mechanism for enforcing security, providing a hierarchical name space in which agents can be referenced and allows agents to address each other locally.

TCP/IP, email: - These are the transport mechanisms through which agents communicate with each other. Agents move between sites in state-oriented fashion by issuing the mobility commands like agent_jump etc.

The agents are executed with the help of interpreters and use the facilities provided by the server to migrate from machine to machine and to communicate with other agents. Agents migrate using the agent_jump command. The agent_jump command captures the internal state of the agent and sends this information to the destination machine. The server on the destination machine loads the appropriate interpreter for the agent, restores the migrated agent’s state information into this execution environment, and resumes the agent’s execution at the statement immediately after the agent_jump. The agent is now on the destination machine and can interact with that machine’s resources without any further network communication.

Agent TCL appears to be the most flexible architecture, since it supports state-oriented migration, multiple languages and networking protocol. It also support security mechanisms but it lacks in constraining agent execution because of which resources are directly accessible via TCL and are freely available for use by agents. It is inefficient than other interpreted languages. It lacks object-oriented features, thus making it difficult to write and debug large scripts.

2. TACOMA

Tromoso And Cornell Moving Agents (TACOMA) [35] is primarily concerned with providing operating system support for agents. TACOMA considers agents, either stationary or mobile, to be computational unit of the system. Each agent has three storage mechanisms: Folder, Filling cabinets and Briefcases as shown in Fig. 2.23. Folder is the essential unit of data that is accessible by an agent. Filling cabinets are stationary data repositories in which folders can be stored. Briefcases are containers that agents carry.
When an agent migrates between sites, state information is stored inside a folder called \textit{DATA}, and the code for the agent itself is stored within another folder called \textit{CODE} and both are within the same briefcase associated with that particular agent. At the receiving site, the code of the agent is extracted from the \textit{CODE} folder and executed; the agent must look in the \textit{DATA} folder for its data and state information.

TACOMA's data storage model is very flexible and the actual way in which information is transferred between agents and servers is beautifully simple. But at higher level programming models where state is invisible to the programmers generates the necessity of automatic state capture. It also suffers from a stateless migration policy.
b) Multi Agent Framework For Agent-Based Focused Crawler

The Multi-Agent framework [33, 36] employs various software agents namely- User Agent, Creator Agent, Matcher Agent and Lexical Agent as shown in Fig. 2.24. All these agents cooperate and coordinate with each other with a common goal to assist the crawler such that the search results are rendered more focused and relevant.

![Multi Agent Framework for Agent Based Focused Crawling](image)

**Fig. 2.24: Multi Agent Framework for Agent Based Focused Crawling**

The Sequence Interaction diagram for the framework representing the interactions between the various active agents along the time-line axis is also shown in Fig 2.25. The User Agent (UA) accepts the keyword from the user and asks the Matcher Agent (MA) to
search the lexical database and return back the entries corresponding to different contexts the keyword can be associated with, along with the URLs at which the information for that particular keyword-context pair can be found.

Once, the user has selected the required context from the list of available contexts, the User Agent provides the corresponding URL seed to the crawler which downloads the web pages and their .TOC. The downloaded web pages are presented to the user as the result of the search query and the .TOCs are used by the Lexical Agent (LA) to maintain and update the lexical database. Since the search has been pruned and restricted only to the documents with desired context, the resulted web pages are more relevant and focused.
2.5.1.5 Distributed Crawlers

In order to download the hundreds of millions of web pages indexed, highly efficient crawling systems are needed [101]. Distributed crawlers were designed keeping in view the following design issues:

- **Full Distribution**: In order to achieve significant advantages in terms of programming, deployment, and debugging, a parallel and distributed crawler should be composed of identically programmed agents, may be distinguished by a unique identifier only. Each of these agents must be executed in a fully distributed fashion, that is, no central coordinator should exist. Full distribution is instrumental in obtaining a scalable, easily configurable system that has no single point of failure.

- **Balanced locally computable assignment**: The distribution of URLs to agents is an important problem, and is directly related to the efficiency of the distributed crawling process. The distribution of URLs should be balanced, that is, each agent should be responsible for approximately the same number of URLs. In case of heterogeneous agents, the number of URLs should be proportional to the agent’s available resources (such as memory, hard disk capacity etc.).

- **Scalability**. The number of pages crawled per second should grow linearly with the number of agents.

- **Politeness**. A suitable delay should be introduced between two subsequent requests to the same host.

- **Fault tolerance**. A distributed crawler should continue to work under crash faults, that is, when some agents abruptly die. When an agent crashes, the remaining agents should continue to satisfy the "Balanced locally computable assignment" requirement, this means, that URLs of the crashed agent will have to be redistributed.

In the following sections the description of some of the prevalent distributed crawlers is given.
a) UbiCrawler

UbiCrawler [26] is composed by several agents that autonomously coordinate their behavior in such a way that each of them scans its share of the web. An agent performs its task by running several threads, each dedicated to the visit of a single host. More precisely, each thread scans a single host using a breadth-first visit. External links are dispatched to the right agent, which puts them in the queue of pages to be visited. Thus, the overall visit of the web is breadth-first, but as soon as a new host is met, it is entirely visited again in a breadth-first fashion.

An important advantage of per-host breadth-first visits is that DNS requests are infrequent. Web crawlers that use a global breadth-first strategy must work around the high latency of DNS servers. This is usually obtained by buffering requests through a multithreaded cache. Similarly, it was suggested that no caching is needed for the robots.txt file required by the Robot Exclusion Standard. Such file can be downloaded when a host visit begins.

Assignments of hosts to agents takes into account the mass storage resources and bandwidth available at each agent. This is currently done by means of a single indicator, called capacity, which acts as a weight used by the assignment function to distribute hosts. Under certain circumstances, each agent gets a fraction of hosts proportional to its capacity. Even if the number of URLs per host varies wildly, the distribution of URLs among agents tends to even out during large crawls. Finally, an essential component of UbiCrawler is a reliable failure detector, which uses timeouts to detect crashed agents. Reliability refers to the fact that a crashed agent will eventually be distrusted by every active agent. The failure detector is the only synchronous component of UbiCrawler i.e. the only component using timings for its functioning, all other components interact in a completely asynchronous way.

b) High Performance Distributed Crawler

Vladislav Shkapenyuk and Torsten Suel [27] has suggested a high performance distributed crawler architecture wherein the whole process is partitioned into two main
components, referred to as *crawling application* and *crawling system* as shown in Fig 2.26. The crawling application decides what page to request next given the current state and the previously crawled pages, and issues a stream of requests (URLs) to the crawling system. The crawling system (eventually) downloads the requested pages and supplies them to the crawling application for analysis and storage. The crawling performs several tasks such as robot exclusion, speed control, and DNS resolution, while the crawling application implements crawling strategies such as "breadth-first" or "focused".

![Fig. 2.26: Basic Structure of High Performance Distributed Crawler](image)

The crawling system itself consists of several specialized components, in particular a *crawl manager*, one or more *downloaders*, and one or more *DNS resolvers*. All of these components, plus the crawling application, can run on different machines (and operating systems) and can be replicated to increase the system performance. The crawl manager is responsible for receiving the URL input stream from the applications and forwarding it to the available downloaders and DNS resolvers while enforcing rules about robot exclusion and crawl speed. A downloader is a high-performance asynchronous HTTP client capable of downloading hundreds of web pages in parallel, while a DNS resolver is an optimized
stub DNS resolver, which forwards queries to local DNS servers. The complete architecture of a High Performance Distributed Crawler is given in Fig 2.27.

The goal of the crawler manager is to download pages in approximately the order specified by the application. Reordering was employed to maintain high performance without putting too much load on any particular web server. After loading the URLs of a requested file, the manager queries the DNS resolvers for the IP addresses of the servers, unless a recent address is already cached. The manager then requests the file robots.txt in the web server's root directory, unless it already has a recent copy of the file. The robot files are written to a separate directory from the other data so they can be accessed and parsed by the manager later.

The downloader component fetches files from the web by opening up to 1000 connections to different servers, and polling these connections for arriving data. Data is then collected into files located in a directory determined by the application and is accessible via NFS. The way pages are assigned to these data files is unrelated to the structure of the request files sent by the application to the manager. Thus, it is up to the
application to keep track of which of its URL requests have been completed. It was reported that DNS lookups generate a significant number of additional frames of network traffic, which may restrict crawling speeds due to limited router capacity.

The **crawling application** is a breadth-first crawl starting out at a set of seed URLs sent to the crawler manager. The application then parses each downloaded page for hyperlinks, checks whether these URLs have been encountered before, and if not, sends them to the manager in batches of a few hundreds or thousands. The downloaded files are then forwarded to a storage manager for compression and storage in a repository.

2.5.1.6 Mobile Crawler

The mobile crawler moves to the data sources i.e. servers before the actual crawling process is started. After accessing a resource, mobile crawlers move on to the next server. It may also carry the crawling result in the memory and return to the originating node. The role of mobile crawlers and the decentralized data retrieval architecture as established by mobile crawlers is shown in Fig 2.28.

![Fig. 2.28: Mobile Crawler](image_url)
The main advantage of this approach is that it allows us to distribute crawling functionality within a distributed system such as the Web. Fig 2.29 depicts the detail architecture of mobile crawler system.

Fig. 2.29: Architecture of Mobile Web Crawling
The Architecture is composed of two modules: the Mobile crawler runtime environment and the Application framework architecture. The Mobile crawler runtime environment supports the execution of crawlers at remote locations, and the application framework architecture, supports the creation and management of mobile crawlers and provides application-oriented functionality, e.g., database connectivity.

Mobile crawlers allow execution of code (i.e. crawlers) on remote systems which are basically web servers. In view of the heterogeneity of the web servers, a crawler generic runtime environment was developed that can be installed easily on any host web server. Thus, any host which installs this runtime environment becomes part of mobile crawling system and is able to retrieve and execute mobile code.

The mobile code architecture consists of two main components. The first component is the communication subsystem which primarily deals with communication issues and provides an abstract transport service for all system components. The second and more important component is the virtual machine which is responsible for the execution of crawlers. In fact the combination of both components is called as the runtime environment for mobile crawlers and provides the functionality necessary for the remote execution of migrating code.

The application framework architecture creates and manages the mobile crawlers and also controls their migration to appropriate locations. Furthermore, it not only analyzes and persistently stores the retrieved data, but also serves as a database front-end, which is responsible for the transformation of the retrieved data into a format compatible to the underlying application.

2.5.1.7 Migrating Crawler

Odysseas et. al. [38, 94, 95] observed that the traditional centralized crawling model suffers from the following limitations:

- The task of processing the crawled data introduces a vast processing bottleneck at the search engine.
- The attempt to download thousands of documents per second creates a network and a DNS lookup bottleneck.
- Documents are usually downloaded by the crawlers in uncompressed form which increases the network bottleneck.

The authors also find that traditional centralized crawling cannot effectively catch up with the dynamic web. So, a novel architecture called UCYMicra [38] was developed. The UCYMicra Crawling System consists of three subsystems: The Coordinator Subsystem, The Mobile Agents Subsystem and a Public Search Engine as shown in Fig 2.30.

![Fig. 2.30: Migrating Crawler](image)

The coordinator subsystem resides at the search engine side and is responsible for maintaining the search database, providing online registration for new Web sites to participate in UCYMicra, and Administering the Mobile Agents Subsystem. The Mobile Agent Subsystem is responsible for crawling the Web. It consists of two categories of mobile agents: The Migrating Crawlers and the Data Carries. The former are responsible
for on-site crawling and monitoring of remote Web servers. Furthermore, they process the crawled pages, and send the results back to the coordinator subsystem for integration in the search engine's database. The latter are responsible for transferring the processed and compressed information from the Migrating Crawlers back to the Coordinator subsystem. The Public Search Engine is responsible for executing user queries on the database maintained by the Coordinator subsystem. Figure 2.31 shows UCYMicra at work.

![Diagram](image)

**Fig. 2.31: UCYMicra at Work**

Powered by their inherent mobile capabilities, the Migrating Crawlers can perform the following tasks:

- **Be dispatched** to a newly registered web server that will participate in UCYMicra.
- **Crawling**: A Migrating Crawler can perform a complete local crawling (either through HTTP or the file system).
• **Processing:** Keywords are extracted from crawled documents, and are ranked based on their visual properties (font and color), position and occurrence frequency, in order to locally create a keyword index of the web server contents.

• **Data transmission:** The index is transmitted to the Coordinator subsystem by the Data Carriers. There it is integrated into the search database.

• **Monitoring:** The Migrating Crawler can detect changes on the Web server contents. Detected changes are instantly processed and transmitted to the Coordinator subsystem.

• **Real time upgrades:** New code for performing any of the above tasks can be easily deployed since UCYMicra’s crawling architecture is based on Java.

2.5.1.8 Parallel Crawlers

Many search engines often run multiple processes in parallel to perform the downloading task, so that download rate is maximized [93, 103]. This type of crawler is known as a parallel crawler. A discussion on the prevalent parallel crawlers is given below:

a) **Parallel Crawler**

In Fig 2.32 the general architecture of a parallel crawler given by Junghoo Cho et al. is illustrated [39]. It consists of multiple crawling processes, referred as C-procs. Each C-proc performs the basic tasks that a single-process crawler conducts. It downloads pages from the Web, stores the pages locally, extracts URLs from them and follows links. Depending on how the C-procs split the download task, some of the extracted links may be sent to other C-procs.

The C-proc's performing these tasks may be distributed either on the same local network or at geographically distant locations. Based on the locations of C-proc's, a parallel crawler can be categorized as follows:

- **Intra-site parallel crawler:** When all C-procs run on the same local network and communicate through a high speed interconnect (such as LAN), it is called an *intrasite parallel crawler*. In this case, all C-procs use the same local network.
when they download pages from remote web sites. Therefore, the network load from C-procs is centralized at a single location where they operate.

- **Distributed crawler:** When a crawler’s C-procs run at geographically distant locations connected by the Internet (or a wide area network), it is called a distributed crawler. A distributed crawler can disperse and even reduce the load on the overall network.

Fig. 2.32: General Architecture of Parallel Crawler

When multiple C-procs download pages in parallel, different C-procs may download the same page multiple times. In order to avoid this overlap, C-procs need to coordinate with
each other on what pages to download. This coordination can be done in one of the following ways:

- **Independent**: At one extreme, C-procs may download pages totally independently, without any coordination. That is, each C-proc starts with its own set of seed URLs and follows links without consulting with other C-procs.
- **Dynamic assignment**: When there exists a central coordinator that logically divides the Web into small partitions (using a certain partitioning function) and dynamically assigns each partition to a C-proc for download, it is called *dynamic assignment*.
- **Static assignment**: When the Web is partitioned and assigned to each C-proc before the start of a crawl, we call it *static assignment*. In this case, every C-proc knows which C-proc is responsible for which page during a crawl and the crawler does not need a central coordinator.

b) **Parallel Crawler based on Augmented Hypertext Documents (PARCAHYD)**

A bottleneck at the document level was found and a novel architecture [40, 41] was proposed. The work was carried out in the following area:

- to develop strategies to crawl only the relevant pages
- to design architectures for parallel crawlers
- restructuring of hypertext documents

For achieving above said goals it was proposed that the links contained within a document should become available to the crawler before an instance of crawler starts downloading the documents itself, the downloading of its linked documents then can be carried out in parallel by other instances of the crawler. Therefore it was proposed [41] that meta-information in the form Table Of Links (TOL) consisting of the links contained in a document be provided and stored external to the document in the form of a file with the same name as document but with different extension (say .TOL). This one time extraction of TOL can be done at the time of creation of the document. The algorithm for the extraction of TOL from hypertext documents was also provided [41].
Firstly, the document retrieval system was divided into two parts: the crawling system and the hypertext (augmented) documents system. The augmented hypertext documents provide a separate TOL for each document to be downloaded by the crawling process. Once the TOL of a document becomes available to the crawler, the linked documents, housed on external sites, can be downloaded in parallel by the other instances of the crawler. Moreover, the overlapped downloading of the main documents along with its linked documents on the same site also becomes possible.

At the second stage, the crawling system was divided into two parts: Mapping Process and Crawling Process. The mapping process (See Fig. 2.33) resolves IP addresses for a URL and Crawling Process downloads and processes documents.

Fig. 2.33: The Mapping Process
The mapping process consists of the following functional components:

**URL-IP Queue:** It consists of a queue of unique seed URL-IP pairs. The IP part may or may not be blank. It acts as an input to the Mapping Manager.

**Database:** It contains a database of downloaded documents and their URL-IP pairs. The structure of its table consists of the following fields:

- URL
- IP-Address
- Document-ID
- Length
- Document

**Resolved URL-Queue:** It stores URLs which have been resolved for their IP addresses and acts as an input to the Crawl Manager.

**URL Dispatcher:** This component reads the database of URLs and fills the URL-IP Queue. It may also get initiated by the user who provides a seed URL in the beginning. It sends a signal: *Something to Map* to the Mapping manager.

**DNS Resolver:** Generally the documents are known by the domain names of their servers. The name of the server must be translated into an IP address before the crawler can communicate with the server [45]. The internet offers a service that translates domain names to corresponding IP addresses and the software that does this job is called the Domain Name System (DNS). The DNS resolver uses this service to resolve the DNS address for a URL and returns it back to the calling URL Mapper. It then updates the database for the resolved IP address of URL.

**MapConf.Txt:** It is a mapper configuration file which is used by the Mapping Manager to load the initializing data.

**Mapping Manager:** This component reads MapConf.txt After receiving the signal *Something to Map*, it creates multiple worker threads called *URL Mapper*. It
extracts URL-IP pairs from the URL-IP Queue and assembles a set of such pairs called URL-IP set. Each URL-Mapper is given a set for mapping.

**URL Mapper:** This component gets a URL-IP set as input from the Mapping Manager. It examines each URL-IP pair and if IP is blank then the URL is sent to the DNS Resolver. After the URL has been resolved for its IP, it is stored in the Resolved URL Queue. It sends a signal *something to crawl* to the Crawl Manager.

The crawling process as shown in Fig 2.34 consists of the following functional components:
**WorkConf.txt:** It is a worker configuration file which is used by the Crawl Manager to load the initializing data.

**Crawl Manager:** This component waits for the signal *something to crawl*. It reads the WorkConf.txt and as per the specifications stored in the file, it creates multiple worker threads named as Crawl Workers. Sets of resolved URLs from Resolved URL Queue are taken and each worker is given a set.

**Crawl Worker:** It maintains two queues: MainQ and LocalQ. The set of URLs received from crawl manager is stored in MainQ. It downloads the documents as per the algorithm given below:

Crawl Worker ()
Begin
    Store the URL set in MainQ;
    While (MainQ is not empty)
        Begin
            Pick up a URL;
            Identify its protocol;
            Download robot.txt;
            If unable to download
                Begin
                    Set IP part as blank;
                    Store URL in Document and URL Buffer;
                    Signal (something to update);
                End;
            Else
                Begin
                    Read robot.txt;
                    Download TOL;
                    Segregate the internal and external Links;
                    Add the URL and internal Links to LocalQ;
                    Store the External Links in the Document and URL buffer;
                    Signal (something to update);
                End;
            End;
        End;
    While (LocalQ is not empty)
        Begin
            Pickup a URL from LocalQ;
            Download document;
            Store the document and its URL in Document and URL Buffer;
            Signal (something to Update);
        End;
    Signal (request processed);
End.
Document and URL Buffer: It is basically a buffer between the crawl workers and the Update Database Process. It consists of following three Queues:

1. External Links queue (ExternLQ): This queue stores the external links.
2. Document-URL queue (DocURLQ): This queue stores the documents along with there URLs.
3. Bad-URL queue (BadURLQ): This queue stores all the resolved Bad-URLs.

Update Database: This process waits for the signal something to update and on receiving the same, updates the database with the contents of the Document and URL Buffer. In order to use storage efficiently, each document is compressed using zlib [42] algorithm.

Each crawl worker independently downloads documents for the URL set received from crawl manager. Since all workers use different seed URLs, it was expected that there will be minimum overlap of downloaded pages. Thus, the architecture requires no coordination overheads among the workers rendering it to be highly scalable system.

For retrieving the pages, the crawler must adopt an efficient search strategy to traverse the site tree. Some of the popular web search strategies employed by a crawler are given in following section.

2.6 WEB SEARCH STRATEGIES

Some of the popular strategies are given below:

- **Breadth-First Search**: In breadth-first search [19] crawler starts with a starting page and follows all URLs residing in the page. After exploring all URLs present in the page it proceeds to the next level and explores all siblings at that level and so on. In fact, starting with the current page it explores all pages that it can reach by using only one hyperlink from the original page as shown in Fig. 2.35.

  *Note: The red line indicates the traverse path across the site tree.*
This process is repeated level by level until no URLs are found. When no more URLs can be located, the search may jump to a new starting point.

- **Depth-First Search**: The process of search goes to process nodes of a site tree as shown in Fig. 2.36.

Child extracts its links and stores them in a stack. Having processed a page, it picks up a link from stack i.e. first child, and visits it. The process is repeated till no links are left in the stack. Thus, as compared to breadth first search where the siblings are
visited first, in depth first search, the childs are searched before other URLs of same page.

- **Fish-Search**: In Fish Search [61], crawler fetches web pages by a query specified by user. The key principle of the algorithm is: “It takes as input a seed URL and a search query, and dynamically builds a priority list (initialized to the seed URL) of the next URLs to be explored.” At each step the first URL is popped from the list and processed. As each document's text becomes available, it is analyzed by a scoring component evaluating whether it is relevant or irrelevant to the search query (1-0 value) and, based on that score, a heuristic decides whether to pursue the exploration in that direction or not. Whenever a document source is fetched, it is scanned for links. The nodes pointed to by these links (denoted "children") are each assigned a depth value. If the parent is relevant, the depth of the children is set to some predefined value. Otherwise, the depth of the children is set to be one less than the depth of the parent. When the depth reaches zero, the direction is dropped and none of its children is inserted into the list.

- **Shark Search**: Shark Search algorithm [62] is an improvement over fish-search algorithm. It overcomes some limitations of fish-search. One improvement is using a fuzzy score to replace the binary (relevant/irrelevant) valuation of document relevance, i.e., a score between 0 and 1 (0 for no similarity whatsoever, 1 for perfect "conceptual" match) rather than a binary value. This improvement has a direct impact on the priority list. The "fuzzy" relevance score gives the child an inherited score, thus preferring the children of a node that has a better score. This information can be propagated down the descendant chain, thus boosting the importance of the grandchildren of a relevant node over the grandchildren of an irrelevant node. A more significant improvement consists of refining the calculation of the potential score of the children not only by propagating ancestral relevance scores deeper down the hierarchy, but also by making use of the meta-information contained in the links to the documents. It uses the hints that appear in the parent document, regarding a child. The anchor text of the link is the author's way to hint as to the subject of the linked document. A surfer on the Web, encountering a page with a large amount of links, will use the anchor text of the links in order to decide how to proceed.
2.7 REVIEW SUMMARY

A critical look at the available literature indicates the following issues need to be addressed towards building an effective Parallel Migrating Crawler:

- Parallel instances running on same host machine leads to Network congestion problem.
- The parallel instances may download duplicate and unfiltered information thus brings redundant and irrelevant documents.
- Existing agent system carries the recently downloaded documents with it while travelling across the web. Thereby not only generating unnecessary load on the internet but also does not update the search engine repository containing the stale information.
- Page change frequency is another important issue i.e. how often pages change and how often it would revisit the server/website in order to maintain the page up to date.
- Nearest server selection is an issue that need to address so that the data, lying on multiple sites, is downloaded from nearest server.
- The code of agent migrated to a remote site needs to secure and the documents downloaded thereof also need to be secured against network attack during the transmission.

A detailed discussion on the models for crawler revisit frequency is given in next chapter.