CHAPTER-2
SEARCH ENGINES AND WEB CRAWLERS

2.1 INTRODUCTION

In the present age the Internet has become an integral part of human life and, out of 7.02 billion population of the world 2.41 billion people (34.3%) make use of Internet [1]. From .36 billion in 2000, the number of Internet users has increased to 2.41 billion in 2012 that makes an increase of 566.4% in just 12 years. In Asia alone, around 1.07 billion people use Internet that makes approximately 45% of worldwide Internet users. In India itself approximately 0.137 billion people use Internet. It is not far away when one will start feeling that life is incomplete without Internet. Same or higher growth rate is expected in future too.

During the last few years, the web has evolved from a handful of pages to billions of diverse objects [22]. The consequence of the popularity of the web as a global information system is that it is flooded with a large amount of data and information. It is estimated that maximum web coverage of any popular search engine is not more than 16% of the current web size. Even the first Google index in 1998 had 26 million pages only, which has increased to three billion in 2012. In 2009 Google had 146 million users monthly that has increased to 931 million in 2010 and one billion in 2011. The web is a context in which traditional information retrieval methods [23] are challenged, as the coverage of modern search engines is limited. Hence finding useful information from the web is often a tedious and frustrating task, and a big challenge for designing efficient modern retrieval methods [24, 25]. Also, the distribution of quality is much distorted, and interesting pages are sparsed in comparison with the rest of the contents.

Due to the business state of affairs only very little literature is available about search engines. One of the first papers in this area introduced the architecture of the World Wide Web Worm (one of the first search engines for the web) was published in 1994. Between 1994 and 1997, the first experimental search engines were followed by larger commercial engines such as WebCrawler, Lycos, Altavista, Infoseek, Excite and HotBot. However, there is very little information available about these search engines and their underlying
technology. Only few papers about architectural aspects of WebCrawler, Lycos and Google project are publicly available.

2.2 THE INTERNET

The Internet [26, 27] is a world-wide broadcasting capability, a mechanism for information dissemination, and a medium for collaboration and interaction between individuals and their computers irrespective of geographic location. The Internet has revolutionized the world of computer and communications like never before. The invention of the telegraph, telephone, radio, and computer set the stage for this unprecedented integration of capabilities. The Internet, as no other medium, has given a global outlook to an individual. Today, it is the Universal source of information. Internet is the most democratic of all the mass media. With a very low investment, anyone can have a web page in Internet. This way, almost any business can reach a global market, directly, fast and economically, no matter what be the size or its location. With a very low investment almost anyone who can read and write can have access to the web.

2.2.1 ORIGIN OF INTERNET

ARPANET [27] began in 1969, and has become the principal archive of information about the Internet. Internet was based on the idea that there would be multiple independent networks of rather arbitrary design, beginning with the ARPANET as the pioneering packet switching network, but soon to include packet satellite networks, ground-based packet radio networks and other networks. The present Internet embodies a key underlying technical idea, namely that of open architecture networking.

In an open-architecture network [27], the individual networks may be separately designed and developed. Each may have its own unique interface which may be offered to users and/or other providers, including Internet providers. Each network can be designed in accordance with the specific environment and user requirements of that network. There are generally no constraints on the types of network that can be included or on their geographic scope, although certain pragmatic considerations will dictate the sense to offer.
The idea of open-architecture networking was first introduced by Kahn [27] shortly after having arrived at DARPA in 1972. At that time, the program was called "Internetting". One of the more interesting challenges was the transition of the ARPANET host protocol from NCP to TCP/IP. It was an approach to develop a new version of the protocol to meet the needs of an open-architecture network environment. This protocol is called the *Transmission Control Protocol/Internet Protocol (TCP/IP)*. The basic ground rules that were critical to design an early network are given below [27]:-

- Each distinct network would have to stand on its own and no internal changes should be required to any such network to connect it to the Internet.
- Communications would be on a best effort basis. If a packet didn't make it to the final destination, it would shortly be retransmitted from the source.
- Black boxes would be used to connect the networks; these would later be called gateways and routers. There would be no information retained by the gateways about the individual flows of packets passing through them, thereby keeping them simple and avoiding complicated adaptation and recovery from various failure modes.
- There would be no global control at the operations level.

The key design issues that were needed to be addressed are given below [27]:-

- Algorithms: To prevent lost packets from permanently disabling communications and enabling them to be successfully retransmitted from the source.
- Pipelining: To enroute multiple packets from source to destination at the discretion of the participating hosts, if the intermediate networks allowed it.
- Gateway functions: To allow forwarding packets appropriately. This includes interpreting IP headers for routing, handling interfaces, breaking packets into smaller pieces, etc.
- End-end checksums: To reassemble packets from fragments and detection of duplicates.
- Others: Need for global addressing, techniques for host to host flow control and interfacing with the various operating systems.
A major initial motivation for both the ARPANET and the Internet was resource sharing. Widespread development of LANs, PCs and workstations allowed the growth of the Internet in 1980s. A major shift occurred as a result of the increase in scale of the Internet and its associated management issues like, to make it easy for people to use the network, hosts were assigned names, so that it was not necessary to remember the numeric addresses. Since there were a fairly limited number of hosts, so it was feasible to maintain a single table of all the hosts and their associated names and addresses.

2.2.2 CURRENT SCENARIO

With the technological evolution that began with packet switching and the ARPANET etc, current research continues to develop the horizons of the infrastructure along several dimensions such as scale, performance, and higher level functionality. Today Internet is a widespread information infrastructure, and what is now called the Global Information Infrastructure can be characterized by the following [27]:-

- It is a collection of logically linked computers/hosts; which are connected to each other by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions as shown in Figure 2.1.

![Figure 2.1: A View of the Internet](image)
• It is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions, and/or other IP-compatible protocols; and
• It provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure.

2.2.3 CONNECTING WITH INTERNET

Since its beginning in 1969, the Internet has grown from four host computer systems to hundreds of millions. One of the greatest things about the Internet is that nobody really owns it. It is a global collection of networks, both big and small. These networks connect together in many different ways to form the single entity that is known as the Internet, as shown in Figure 2.2.

These networks rely on Network Access Points (NAPs), backbones and routers to talk to each other [28]. A message can leave one computer and travel halfway across the world through several different networks and arrive at another computer in a fraction of a second. Routers are the specialized computers that facilitate to send our messages to their destinations along thousands of pathways. The routers determine where to send information from one computer to another.
A router has the following two major jobs [28]:-

- It ensures that information does not go undesired destinations.
- It ensures that the information reaches to the intended destination.

In other words it not only joins the two different networks, passing information from one to the other, but also protects the networks from one another, preventing the traffic on one from unnecessarily spilling over to the other. Regardless of how many networks are attached, the basic operation and function of the router remains the same. Since the Internet is a huge network consisting of tens of thousands of smaller networks, use of routers is essential.

2.2.4 GOVERNING BODIES

Number of bodies [27, 28] has been assigned the responsibility for Protocols and the parameters that are required to run the Internet. The Internet Society (ISOC) is a professional membership organization of Internet experts that comments on policies and practices and oversees a number of other boards and task forces dealing with network policy issues and, the units operating under it namely Internet Architecture Board (IAB), Internet Engineering Steering Group (IESG), Internet Engineering Task Force (IETF), Internet Research Steering Group (IRSG), Internet Research Task Force (IRTF) and Request for Comments (RFC) Editor.

The IAB is responsible for defining the overall architecture of the Internet, providing guidance and broad direction to the IETF. The IAB also serves as the technology advisory group to the Internet Society and oversees a number of critical activities in support of the Internet. The IESG is responsible for technical management of IETF activities and the Internet standards process. It administers the process according to the rules and procedures which have been ratified by the ISOC. It is directly responsible for the actions associated with entry into and movement along the Internet "standards track" including final approval of specifications as Internet Standards. The IETF is the protocol engineering and development arm of the Internet. The IANA is in-charge of all "unique parameters" of the Internet, including Internet Protocol (IP) addresses. Each domain name is associated with a unique IP address, a numerical name consisting of four blocks of
consisting a maximum of three digits each (e.g. 204.146.46.8) that systems use to direct information through the network.

2.2.5 HYPERTEXT TRANSFER PROTOCOL

Hypertext Transfer Protocol (HTTP) is the protocol used for document exchange in the web. Everything that occurs on the web happens over HTTP transactions. TCP/IP networking and HTTP are the two essential components that make the web to work. HTTP [29, 30] is a client-server protocol by which two machines can communicate over a TCP/IP connection. A HTTP server is a program that sits listening on a machine's port for HTTP requests. A HTTP client opens a TCP/IP [31] connection to the server via a socket, transmits a request for a document, and then waits for a reply from the server. Once the request-reply sequence is completed, the socket is closed. So the HTTP protocol is a transactional one. The lifetime of a connection corresponds to a single request-reply sequence i.e. a transaction.

A. Terminology

The basic terms and relations to HTTP are given below:-

- **Connection**: A transport layer virtual circuit established between two application programs for the purpose of communication.
- **Message**: The basic unit of HTTP communication, consisting of a structured sequence of octets matching the syntax and transmitted via the connection.
- **Request**: An HTTP request message.
- **Response**: An HTTP response message.
- **Resource**: A network data object or service, which can be identified by a Uniform Resource Identifier (URI).
- **Entity**: A particular representation or rendition of a data resource, or reply from a service resource, that may be enclosed within a request or response message. An entity consists of Meta information in the form of entity headers and content in the form of an entity body.
- **Client**: An application program that establishes connections for the purpose of sending requests.
• **User agent**: The client, which initiates a request i.e. browsers, editors, spiders, or other end user tools.

• **Server**: An application program that accepts connections in order to service requests by sending back responses.

• **Cache**: A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.

Any client or server may include a cache, though a server cannot use a cache while it is acting as a tunnel. Any given program may be capable of being both a client and a server; use of these terms refers only to the role being performed by the program, rather than to the program's capabilities in general. Likewise, any server may act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

**B. Internet Protocols**

Every Computer (or machine) on the Internet has a unique identifying number, called an Internet Protocol Address. Internet Protocol (IP) is the language that computers use to communicate over the Internet [31]. The IP address is a 32 bit integer. A typical IP address looks like 217.26.60.136. IP addresses are normally expressed in decimal format as a *dotted decimal number* making it easier for user to remember. However computers communicate in binary form, and the same IP address in binary is 11011001.00011010.00111100.10001000

The four numbers in an IP address are called *octets*, and adding all the positions together sums upto 32 making IP addresses of *32-bit numbers*. As each of the eight positions can have two different states (0 or 1), the total number of possible combinations per octet is $2^8$ or 256. So each octet can contain any value between zero and 255. Out of the almost 4.3 billion possible combinations, certain values are restricted from use as typical IP addresses. For example, the IP address 0.0.0.0 is reserved for the default network and the address 255.255.255.255 is used for broadcasts. The octets are used to create classes of IP addresses that can be assigned to a particular business, government or other entity based on size and need. The octets are split into two sections: *Netid and Hostid*. The Netid
section always contains the first octet and is used to identify the network to which a computer belongs. Hostid identifies the actual computer on the network.

2.3 DOMAIN NAME SYSTEM

If somebody wants to send a message it is necessary to include the destination address, but people prefer to assign machines pronounceable, easily remembered names (host names). In 1983, the University of Wisconsin created the Domain Name System (DNS), which maps text names to IP addresses automatically [28]. This way, users only need to remember the name, for example http://www.shobhituniversity.ac.in, instead of Shobhit University’s IP address. These logical names also allow independence from knowing the physical location of a host. A host may be moved to a different network, while the users continue to use the same logical name.

The Domain Name System (DNS) is a distributed database [31] used by TCP/IP applications to map between hostnames and IP addresses. It is also used to provide electronic mail routing information, as shown in Figure 2.3. Each site (for example university department, company, or departments within a company) maintains its own database of information and runs a server program that other systems across the Internet can query. The DNS provides the protocol, which allows clients and servers to communicate with each other.

![Figure 2.3: TCP/IP Working](image-url)
The system accesses the DNS through a *resolver* (see Figure 2.4). The resolver gets the hostname and returns the IP address or gets an IP address and looks up for a *hostname*. The resolver returns the IP address before asking the TCP to open a connection or sending a datagram using UDP.

![Diagram of DNS working](image)

**Figure 2.4:** Working of DNS

### 2.3.1 DNS ORGANIZATION

The domain name system uses a hierarchical naming scheme [31] known as domain names (similar to UNIX file system tree). The root of the DNS tree is a special node with a null label, and the name of each node (except root) has a maximum of 63 characters. 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 or a dot (".") to separate the labels (individual sections of a name might represent sites or a group, but the domain system simply calls each section a label). The difference between the UNIX file system and the tree of the DNS is that the DNS starts on the ground and go up to the root. Writing them in this order makes it possible to compress messages that contain multiple domain names.
The domain name "shobhituniversity.ac.in" contains three labels "shobhituniversity", "ac", and "in" as shown in Figure 2.5.

Any suffix of a label in a domain name is also called a domain. In this example, the lowest level domain is “shobhituniversity” (the domain name for Shobhit University, India), the second level domain is “ac” (the domain name for academic organizations), and the top-level domain here is in (the domain name for India). 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 [31] as given below:

(i) \textit{arpa} is a special domain used for address-to-name mapping.
(ii) The seven 2/3 character domain names (generic (organizational) domains), e.g. \textit{com, edu, gov, ac} etc.
(iii) 2/3 character domains are based on the country codes (\textit{in, uk, usa, aus}). These are called the country (the geographical) domains.

Top level domain names (or first-level domain names) include .com, .org, .net, .edu, .ac or .gov. Within every top level domain there is a huge list of second-level domains. Every name in the .ac top level domain must be unique. The left-most word, like www, is the
host name. It specifies the name of a specific machine in a domain. A given domain can, potentially contain millions of host names as long as they are all unique within that domain.

DNS servers accept requests from programs and other name servers to convert domain names into IP addresses. When a request comes in, the DNS server can do one of four things with it [28]:-

(i) It can answer the request with an IP address because it already knows the IP address for the requested domain.
(ii) It can contact another DNS server and try to find the IP address for the name requested. It may have to do this multiple times.
(iii) It can say, "I don't know the IP address for the domain you requested, but can refer the IP address of a DNS server that knows."
(iv) It can return an error message because the requested domain name is invalid or does not exist.

For URL http://www.shobhituniversity.ac.in, into the browser, the browser contacts a DNS server to get the IP address. A DNS server would start its search for an IP address by contacting one of the root DNS servers. The root servers know the IP addresses for all of the DNS servers that handle the top-level domains (com, net, ac etc.). The name server then sends a query to the ac DNS server asking it if it knows the IP address for the site. The DNS server for the ac domain knows the IP addresses for the name servers handling the shobhituniversity domain, so it returns those. The name server then contacts the DNS server for http://www. shobhituniversity.ac.in and asks for the IP address of http://shobhituniversity.ac.in which returns the IP address to DNS server, which in turn returns it to the browser, which can then contact the server for http://www. shobhituniversity.ac.in to get a web page.

One of the keys to make this work is redundancy. There are multiple DNS servers at every level, so that if one fails, there are others to handle the requests. The other key is caching. Once a DNS server resolves a request, it caches the IP address it receives. Once it has made a request to a root DNS server for any ac domain, it knows the IP address for a DNS server handling the ac domain, so it doesn't have to bug the root DNS servers again for that information. DNS servers can do this for every request, and this caching
helps to keep things from bogging down. Even though it is totally invisible, DNS servers handle billions of requests every day and they are essential to the Internet's smooth functioning.

### 2.4 WORLD WIDE WEB

The World Wide Web [2, 32, 33] (WWW or web) is a web of hyperlinked repository of trillions of hypertext documents lying in different websites, distributed over far end distant geographical locations as shown in Figure 2.6.

While working with www, first the server-name portion of the URL is resolved into an IP address using the global, distributed Internet database known as the domain name system (DNS). This IP address is necessary to contact and send data packets to the web server. The browser then requests the resource by sending a HTTP request to the web server at that particular address. In the case of a typical web page, the HTML text of the page is requested first and parsed immediately by the web browser, which will then make additional requests for images and any other files that form a part of the page. Having received the required files from the web server, the browser then renders the page onto the screen as specified by its HTML and other web languages. Any images and other resources are incorporated to produce the on-screen web page that the user sees.
Most web pages will themselves contain hyperlinks to other related pages and perhaps to downloads, source documents, definitions and other web resources. Such a collection of useful, related resources, interconnected via hypertext links, is what was dubbed a web of information.

The key factors for the success of the web are its large size and the lack of a centralized control over its contents. Both issues are also the most important source of problems for locating information. The web is a context in which traditional information retrieval methods are challenged, and given the volume of the web and its speed of change, the coverage of modern search engines is relatively small. Moreover, the distribution of quality is very skewed, and interesting pages are scarce in comparison with the rest of the content.

2.4.1 DEVELOPMENT OF WWW

It all began when Tim Berners-Lee (a graduate of Oxford University) got frustrated with the fact that his daily schedule planner, his list of phone numbers, and his documents were stored in different databases on different machines thus making it difficult to access them simultaneously. He set out to fix this problem at CERN (Centre European pour la Recherché Nucleaire - or - European Laboratory for Particle Physics) [27] in 1980.

In March of 1989, Tim Berners-Lee submitted “Information Management: A Proposal” to his superiors at CERN. In a later paper "World-Wide Web: An Information Infrastructure for High-Energy Physics", he mentioned that the motivation for this system arose "from the geographical dispersion of large collaborations, and the fast turnover of fellows, students, and visiting scientists," who had to get "up to speed on projects and leave a lasting contribution before leaving." In his original "Information Management: A Proposal," Berners-Lee described the deficiencies of hierarchical information delivery systems and outlined the advantages of a hypertext-based system. The proposal called for "a simple scheme to incorporate several different servers of machine-stored information already available at CERN." A distributed hypertext system was the mechanism to provide "a single user-interface to many large classes of stored information such as reports, notes, data-bases, computer documentation and on-line systems help." The proposal's main objectives were:-
• 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 of some method of reading text (and possibly graphics) using a large proportion of the display technology in use at CERN at that time.
• 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 following the hyperlinks.
• To use public domain software wherever possible and to interface to existing proprietary systems.
• To provide the necessary software free of charge.

Berners-Lee et al at CERN paved the way for the future development of the web by introducing their server and browser, the protocol used for communication between the clients and the server, Hypertext Transfer Protocol (HTTP), the language used in composing web documents, Hypertext Markup Language (HTML), and the Universal Resource Locator (URL).

2.4.2 WEB SERVERS

Internet is possible through Internet servers. All the machines on the Internet are either servers or clients. The machines that provide services to other machines are servers and the machines that are used to connect those services are clients. There are web servers, e-mail servers, FTP servers etc. serving the need of Internet users all over the world.

When a user connects to http://www. shobhituniversity.ac.in to read a page, sitting at a client's machine. He/she accesses the Shobhit University’s web server. The server machine finds the requested page and sends it to the user. Clients that come to a server machine do so with a specific intent, so clients direct their requests to a specific software server running on the server machine. A server has a static IP address that does not
change very often. On the other hand a home machine that is dialing up through a modem typically has an IP address assigned by the ISP every time a user dials in. That IP address is unique for that session, it may be different for the next time a user dials in. This way, an ISP only needs one IP address for each modem it supports, rather than one for each customer.

In recent years, the web creates many new challenges for information retrieval because of its very large size and heterogeneous nature. Currently there are over thousands of millions of web pages with a doubling life of less than a year. The web pages are extremely diverse, ranging from normal pages to journals of information. Web contains various types of file like HTML, DOC, XLS, JPEG, AVI, PDF etc.

The current graph [34] of the crawlable web has roughly thousand millions nodes, and billions of links. Every page has some number of forward links i.e. out_edges and back_links i.e. in_edges (see Figure 2.7). If a user has downloaded a page, he/she may know all of its forward links but can never know whether all the back links of a particular page have been found. Web pages vary in terms of the number of back links they have. Generally, highly linked pages are more important than pages with few links. PageRank [13, 35] is an attempt to see how good an approximation to importance can be obtained from the link structure.

![Figure 2.7: Link Structure of the Web](image-url)
2.4.3 WEB BROWSER

Web browser (or browser) is a software application used to enable computers users to locate and access web pages. Browsers translate the basic HTML code that allows users to see images, text videos and listen to audios on websites, along with hyperlinks that let travel to different web pages. The browser gets in contact with the web server and requests for information. The web server receives the information and displays it on the computer.

A web browser contains the basic software that user needs in order to find, retrieve, view, and send information over the Internet. This includes software that lets the user:-

- Send and receive electronic-mail (e-mail) messages worldwide nearly instantaneously.
- Read messages from newsgroups (or forums) about thousands of topics in which users share information and opinions.
- Browse the web where one can find a rich variety of text, graphics, and interactive information.

With a web browser, a user views web pages containing text, images, videos, and other multimedia and navigates between them using hyperlinks. Viewing a web page on the World Wide Web normally begins either by typing the URL of the page into a web browser, or by following a hyperlink to that page or resource. The web browser then initiates a series of communication messages, behind the scenes, in order to fetch and display them.

2.5 NEED OF WEB SEARCH ENGINES

The web collectively contains huge information housed in web servers. Since 1990, the web has grown exponentially in size, and as of today it is estimated that it contains at least 14.27 billion publicly accessible/indexable web documents distributed all over the world on thousands of web servers [36]. According to Bal and Nath [37] the web is very dynamic in nature and 40% of its contents change daily. Large size and the lack of a centralized control over its contents are two key factors for the success of web. Both issues are the most important source of problems for locating information. The web is a
context in which traditional information retrieval methods are challenged, and provides the volume of the web, its speed of change. The coverage of modern search engines is relatively small. Moreover, the distribution of quality is very off-centered, and interesting pages are scarce in comparison with the rest of the contents.

It is very difficult to search relevant information from such huge collection of web documents as the web pages are neither organized as books on shelves in a library, nor are completely catalogued at one central location. It is not guaranteed that users will be able to retrieve information even after knowing where to look for information by knowing its URLs as web is constantly changing. Therefore, there is a need to develop efficient information retrieval tools to search the required information from the web.

To help this sheer volume of data available on the web, programs that run off of special websites called search engines; are being designed [38]. A search engine is a coordinated set of programs that is able to read every searchable page on the web, create an index of the information it finds, compare that information to a user's search request (i.e. query), and finally return the results back to the user.

Web search engines [4, 28, 39, 40] work by storing information about many web pages which they retrieve from the web itself. These pages are retrieved by a web crawler; an automated web browser which follows every link it sees. Exclusions can be made by the use of robots.txt. The contents of each page are then analyzed to determine how it should be indexed (for example, words are extracted from the titles, headings, or special fields called meta tags). Data about web pages are stored as an index database for use in later queries.

When a user enters a query into a search engine (normally by using key words), the engine examines its index and provides a listing of best-matching web pages according to its criteria, usually with a short summary containing the document's title and sometimes parts of the text. Most search engines support the use of the boolean operators AND, OR and NOT to further specify the search query. Some search engines provide an advanced feature called proximity search which allows users to define the distance between keywords (i.e. they are separated by N words between them).
The usefulness of a search engine depends on the relevance of the result set, while there may be millions of web pages that include a particular word or phrase, out of which some pages may be more relevant, popular, or authoritative than others. Most search engines employ methods to rank the results to provide the best results first. How a search engine decides which pages are the best matches, and the order the results should be shown in, varies widely from one engine to another. The methods also change with time as Internet usages change and new techniques evolve.

2.6 SEARCH ENGINES

As explained earlier, search engines [4, 39, 41] act as a bridge between web users and web pages. Without search engines, the unlimited source of information stored in web pages remain unexplored for the users. A search engine is a searchable database which collects information from web pages on the Internet, indexes the information and then stores it in a huge database from where it can be quickly searched.

There are several different ways search engines work, but they all perform three basic tasks:-

(i) They search the Internet based on important (key) words.
(ii) They keep an index of the words they find with their location.
(iii) They allow users to look for words or combinations of words in the index.

2.6.1 ARCHITECTURE OF A WEB SEARCH ENGINE

In general, a typical web search engine has three parts a crawler, indexer and query engine as shown in Figure 2.8.

Web search engines [42] employ crawlers to continuously collect web pages from the web. The downloaded pages are indexed and stored in a database. This continuous updation of database renders a search engine more reliable source of right and updated information. Web crawlers are small programs that peruse 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 crawler has to deal with two main responsibilities i.e. downloading the new pages and keeping the freshness of previously downloaded pages. However, good freshness can only be guaranteed by simply revisiting all the pages frequently which is not possible and it puts unnecessary overload on the crawler. With the available bandwidth for conducting crawls which is neither infinite nor free, it is becoming essential to crawl the web in not only scalable but efficient way if reasonable measure of quality or freshness is to be maintained.

The *indexer* extracts all the 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.
• **Full text indexing** is where every word on the page is put into a database for searching. It helps user to find every example in response to a specific name or terminology. A general topic search will not be very useful in the database and one has to dig through a lot of false drops.

• In **keyword indexing** only important words or phrases are placed 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.

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.

### 2.6.2 CHALLENGES IN DATA TRAVERSING BY SEARCH ENGINES

Several challenges faced to traverse the large volume of unstructured web data are:-

• **Distributed data:** Data is distributed widely in the web world, and is located at different sites and platforms. The communication links between computers vary widely, and there is no topology of data organization.

• **High percentage of volatile data:** Documents can be added or removed easily in the web. Changes to these documents may remain unnoticed by users.

• **Large volume:** The growth of data is exponential. It causes scaling issues that are difficult to cope with.

• **Unstructured and redundant data:** The web is not exactly a distributed hypertext. Web pages are not well structured and semantic redundancy can increase traffic.

• **Quality of data:** A lot of web pages do not involve any editorial process which means that data can be false, inaccurate, outdated, or poorly written.

• **Heterogeneous data:** Data on the web are heterogeneous in nature i.e. they are written in different formats, media types, and natural languages.

• **Dynamic data:** Content of web documents change dynamically. The contents can be changed by a program such as hit counter that keep tracks of number of hits.
2.6.3 TYPES OF SEARCH ENGINES

Search engines are good at finding unique keywords, phrases, quotes; and information contained in the full text of web pages. Search engines allow user to enter keywords and then search for them in its table followed by database. The broad categories of search engines based on their working are given below [43, 44]:

- **Crawler based Search Engines:** Crawler based search engines [43] create their listings (that make up the search engine's index or catalog) automatically with the help of web crawlers. It uses a computer algorithm to rank all pages retrieved. Such search engines are huge and often retrieve a lot of information from the web. For complex searches, it allows to search within the results of previous search and enables us to refine search results. Such types of search engines contain full text of web pages they link to. Here, one can find pages by matching words in the pages user wants. The examples of crawler based search engines are Google, Altavista, Teoma, Lycos and AllTheWeb etc.

- **Human powered Directories:** A person who actually creates the website’s listing on the search page, as opposed to a ‘robot’ or ‘spider’ to do this automatically, creates Directories. Human powered directories [43, 44] are built by human selection i.e. they depend on humans to create repository. They are organized into subject categories and classification of pages is done by subjects. Such directories do not contain full text of the web page they link to, and are smaller than most search engines.

A short description and the URL to the website is submitted to the search engine owners. If approved, the search owners then assign the website to an appropriate category or categories within the large search website. Directories often provide much more targeted results than search engines. A search for the directory site looks for matches only within the descriptions submitted, not information found on the web pages. To update the website’s description, we need to submit an online update to the search engine's webmaster. Some examples of human powered directories are Yahoo, MSNSearch, AskJeeves, Excite, Look Smart.

- **Hybrid Search Engines:** Hybrid search engines [43, 44] differ from traditional text oriented and directory based search engines. These search engines typically
favor one type of listing over the other. Many search engines today combine a crawler based search engine with a directory service. All the major search engines like Google are good examples of hybrid search engines.

- **Meta Search Engines:** Meta search engines [43, 44] or sometime called meta crawler accumulate search and screen the results of multiple primary search engines. Unlike search engines, meta crawlers don't crawl the web themselves to build listings. Instead, they allow searches to be sent to several search engines all at once. The results are then blended together onto one page. Examples of meta search engines are Dogpile, Highway61, Mamma etc.

Based on the application for which search engines are used, they can be categorized as follows:-

- **Primary search engines** scan entire sections of the www and produce their results from databases of web page content, automatically created by computers.
- **Business and Services search engines** essentially National yellow page directories.
- **Employment and Job search engines** either provide potential employers access to resumes of people interested in working for them or provides prospective employees with information on job availability.
- **Finance-oriented search engines** facilitate searches for specific information about companies (officers, annual reports etc.).
- **Image search engines** help us search the www for images of all kinds.
- **News search engines** search newspaper’s and news websites archives for the selected information.
- **People search engines** search for names, addresses, telephone numbers and e-mail addresses of the people.
- **Subject guides** are like indexes in the back of a book. They involve human intervention in selecting and organizing resources, so they cover fewer resources and topics but provide more focus and guidance.
- **Specialized search engines** search specialized databases, allow users to enter search terms in a particularly easy way, look for low prices on items they are
interested in purchasing, and even give users access to real, live human beings to answer questions.

2.7 DESIGN ISSUES OF SEARCH ENGINES

Designing a large-scale search engine is a non-trivial task and entails huge challenges [39, 44, 45, 46, 47, 48]. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been carried on them. Furthermore, due to rapid advance in technology and web proliferation, designing a web search engine today is very different from few years ago.

A perfect automated search engine in the current scenario is one that crawls the web quickly and gathers all the documents to keep them up-to-date. Plenty of storage space is required to efficiently store indices or the documents themselves. The magnitude of data that has to be handled on the ever-growing internet includes billions of queries daily. The indexing system of a search engine should be capable of processing huge amount of data by using the space most efficiently and handling thousands of queries per second. The best navigation experience should be provided to the users, in the form of finding almost anything on the web, excluding the junk results with the use of high precision tools, which is the main problem faced by the users.

There are different ways to improve performance of search engines but there are three main characteristics: improving algorithms to traverse the web, using filtering mechanisms for optimizing the user’s results; and improving the user interface for query input. Factors that determine the quality of a search engine are freshness of contents, index quality, search features, retrieval system and user behaviour. In addition to above other important issues are as follows:-

- Diversity of documents: On the web, documents are written in several different languages. Documents need to be indexed in a way which allows it to search for documents written in diverse languages with just one query. Search engines should be able to index documents written in multiple formats, as each file format provides certain difficulties for the search engines.
- **Web user's behaviour**: Web users are very heterogeneous and search engines are used by professionals as well as by laymen. The search engine needs to be enough smart to serve both types of users. Moreover, there is a tendency that users often only look at the results set available without scrolling, and the results which are not on first page are nearly invisible for the general user.

- **Freshness of database**: Search engines find problems in keeping its database up-to-date with the entire web because of its enormous size and the different update cycles of individual websites.

Other characteristics [49] that a large search engine is expected to have are *scalability, high performance, politeness, continuity, extensibility and portability.*

### 2.8 WEB CRAWLERS

Web crawlers have been defined [47, 50, 51, 52, 53, 54] as *“software programs that traverse the world wide web information space by following hypertext links and retrieving web documents by standard HTTP protocol”*. They are the software that automatically retrieves web documents by standard HTTP protocol, either by following hypertext links or other methods. Alternatively, a web crawler is a program [42] that fetches information from www in an automated manner. The objective of a web crawler is to maintain freshness [8, 10, 55, 56, 57] of pages in its collection as high as possible. A web crawler deals 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 robust against crashes, manageable, and considerate of resources and web servers.

#### 2.8.1 GENERAL ARCHITECTURE OF A WEB CRAWLER

As discussed earlier, the web crawlers are small programs that peruse 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. To download a document, it picks up a URL from its seed URLs and depending upon the host protocol, downloads the documents from the web server. The browser parses the document and makes it available to the user [33].

Generally, a crawler starts by placing an initial set of seed URLs [42] in a queue, where all URLs to be retrieved are kept and prioritized (see Figure 2.9). From this queue the crawler extracts a URL, downloads the page, extracts URLs from the downloaded page, and places the new URLs in the queue. This process is repeated and the collected pages are used by a search engine. The browser parses the document and makes them available to the users.

![Figure 2.9: General Architecture of a Web Crawler](image)

The main objective of a web crawler is to maintain freshness of pages in its collection as high as possible or, to keep the average age of pages as low as possible, i.e. the crawler is concerned with how many pages are outdated and how old the local copies of pages are.

To understand the operation of a web crawler one has to look at its generic structure:-
• **Retrieval stage:** The ultimate goal of a crawler is to establish a web index, the crawler has to retrieve the resources which will be part of the index. Typically, the crawler contacts a remote HTTP (Hypertext Transfer Protocol) server, requesting a web page specified by a URL (Uniform Resource Locator) address.

• **Analysis stage:** After a certain resource has been retrieved, the crawler will analyze the resource as per the crawling algorithm. For example, in case the retrieved resource is a web page, the crawler will probably extract hyperlinks and keywords contained in the page.

• **Decision state:** Based on the results of the analysis stage, the crawler has to make a decision to precede the crawling process. For example, the crawler might identify some (or all) of the extracted hyperlinks as being candidates for the index by providing them as new input for the retrieval stage. This will restart the process.

On examining crawlers in the context of the above, the particular differences in crawling strategies are due to different implementations of the stages identified.

### 2.8.2 CHALLENGES IN IMPLEMENTING A CRAWLER

With such huge size and change rate of the web, the crawlers need to address many challenges including the following [58, 59]:-

**A. Which pages to download**

In most cases, the crawler cannot download all pages of the web. Even the most comprehensive search engine currently indexes a small fraction of the entire web. Given this fact, it is important for the crawler to carefully select the important pages and to visit them first, so that the fraction of the web that is visited is more meaningful, and the information collected in the local repository is more useful.

**B. How to refresh pages**

Once the crawler has downloaded a significant number of pages, it has to start *revisiting* the downloaded pages in order to detect changes and refresh the downloaded collection. Because web pages are changing at very different rates, the crawler needs to carefully
decide which pages to revisit and which pages to skip in order to achieve high freshness of pages.

C. To minimize load of websites to visit

When the crawler collects pages from the web, it consumes resources belonging to other organizations. When the crawler downloads page from a site, the site needs to retrieve page from its file system, consuming disk and CPU resources. After this retrieval, the page then needs to be transferred through the network, another resource shared by multiple organizations. Therefore, the crawler should minimize its impact on these resources. Otherwise, the administrator of a website or a particular network may complain and sometimes may completely block to be accessed by the crawler.

D. Parallelizing the crawling process

Due to the enormous size of the web, crawlers often run on multiple machines and download pages in parallel [60]. This parallelization is often necessary in order to download a large number of pages in a reasonable time. Clearly these parallel crawlers should be coordinated properly, so that different crawlers do not visit the same website multiple times. However, this coordination can incur significant communication overhead, limiting the number of simultaneous crawlers.

2.8.3 COST OF CRAWLING

The web crawler is useful for a number of tasks, but comes with a price for the general community. The costs of crawling includes:

- **Network resources utilization**: Since crawlers require considerable bandwidth and operate with a high degree of parallelism during a long period of time.
- **Overloading of web server**: Especially if the frequency of accesses to a given server is too high.
- **Poorly written crawlers**: Which can crash servers or routers, or download pages they cannot handle.
- **Personal crawlers**: If deployed by too many users, can disrupt networks and web servers.
2.8.4 ISSUES WITH CURRENT CRAWLERS

Various issues related with the available crawling strategies are given below:-

A. Scaling

Within the last few years, search engine technology had to scale dramatically to keep up with the growing amount of information available on the web. One of the first web search engines, the world wide web worm, was introduced in 1994 and used an index of 1,10,000 web pages. Big commercial search engines in 1998 claimed to index up to 110 million pages. This is an increase by a factor of 1,000 in four years only. Even the first Google index in 1998 had 26 million pages only, which has increased to three billion in 2012. In 2009 Google had 146 million users monthly that has increased to 931 million in 2010 and more than one billion in 2012. According to [37], the maximum web coverage of any popular search engine is not more than 16% of the current web size. The web is expected to grow further at an exponential speed, doubling its size (in terms of number of pages) in less than a year. The task of processing this vast amount of data creates a bottleneck at search engines side.

B. Efficiency

Since web crawlers generate a significant amount of web traffic, one might think whether all the data downloaded by a crawler are really necessary. The answer to this is almost always no. In the case of specialized search engines, the answer is definitely no, because these engines focus on providing good coverage of specific subject areas only and are not interested in all of the web pages retrieved by their crawlers. Hence, the attempt to download thousands of pages per second creates a network bottleneck.

C. Index Quality

Even if the advances in storage and network capacity can keep pace with the growing amount of information available on the web, it is questionable whether a larger index necessarily leads to better search results. From the user's point of view it doesn't make any difference whether the engine returned 90,000 or 900,000 matches because the huge number of matches is not manageable. Instead of trying to accommodate an estimated
index size of up to one billion pages, more efforts should be made to improve the quality of web indices in order to establish a base for improved search quality.

**D. Network bandwidth**

Documents are usually downloaded by the crawlers in uncompressed form that causes the network bottleneck. In general, compression is not under full facilitation since it is independent of the crawling task and cannot be forced by the crawlers. In addition, for every page to be fetched corresponding to a URL (downloading the page indexed by URL) crawler has to maintain a new HTTP connection every time hence creating a lot of traffic in the network.

**E. Flexibility**

The Crawler should be flexible in nature, to be able to be used in a variety of scenarios like modifications as possible, customizable with lots of options and enables communities to customize it. Flexible in configuring the retrieval, processing and storage of harvested contents, with small indexes it should be able to run with very little memory and be able even scale down to a personal computer.

**F. Low Cost and High Performance**

The system should scale to at least several hundred pages per second and hundreds of millions of pages per run, and should run on low cost hardware. The efficient use of disk access is crucial to maintain a high speed; after the main data structures become too large for main memory.

**G. Robustness**

There are several aspects; first, since the system will interact with millions of servers, it has to tolerate bad HTML, strange server behavior and configurations, and many other odd issues. Secondly, since a crawl may take weeks or months, the system needs to be able to tolerate crashes and network interruptions without losing too much of the data. Thus, the state of the system needs to be kept on disk.

**H. Etiquette and Speed Control**
It is extremely important to follow the standard conventions for robot exclusion (robots.txt), to supply a contact URL for the crawler, and to supervise the crawl. A user needs to be able to control access speed in several different ways. He has to avoid putting too much load on a single server; and can be done by contacting each site only once every second unless specified otherwise. It is also desirable to throttle the speed on a domain level, in order not to overload small domains. There is also a need to control the total download rate of the crawler.

I. Manageability and Reconfigurability

An appropriate interface is needed to monitor the crawl, including the speed of the crawler, statistics about hosts and pages, and the sizes of the main data sets. The administrator should be able to adjust the speed, add and remove components, shutdown the system, force a checkpoint, or add hosts and domains to a “blacklist” of places that the crawler should avoid. After a crash or shutdown, the software of the system may be modified to fix problems, and user may want to continue the crawl using a different machine configuration.

2.9 CRAWLING TECHNIQUES

Several web crawling techniques [61] are in use that differs in their mechanism, implementation and objective. Some prevalent web crawling techniques are given below:-

2.9.1 PARALLEL AND DISTRIBUTED CRAWLERS

Crawling the complete web is a big challenge due to its growing and dynamic nature. As the size of the web is growing it has become imperative to parallelize the crawling process in order to finish downloading the pages in a reasonable amount of time. A single crawling process (even multithreading) is insufficient for large scale engines that need to fetch large amounts of data rapidly. When a single centralized crawler is used, all the fetched data passes through a single physical link. Distributing the crawling activity via multiple processes can help build a scalable, easily configurable system, which is fault tolerant system. Splitting the load decreases hardware requirements and at the same time increases the overall download speed and reliability. Each task is performed in a fully distributed fashion, that is, no central coordinator exists.
In a parallel crawler [60, 61, 62, 63] multiple crawling processes (C-Proc’s) run in parallel to perform the crawling task, which maximizes the download rate as shown in Figure 2.10.

![General Architecture of a Parallel Web Crawler](image)

**Figure 2.10:** General Architecture of a Parallel Web Crawler

Each C-proc performs the basic tasks that a single process’s crawler conducts. Here, all C-Proc's run on the same local network and communicate through a high speed interconnection, and use the same local network to download web pages. The network load from C-Proc's is centralized at a single location where they operate. To improve the quality of downloaded pages and to prevent overlap, the crawling processes need to communicate with each other.

A distributed crawler [61, 64, 65, 66, 67, 68] harnesses the excess bandwidth and computing resources of clients to crawl the web. It is building a decentralized search engine over peer-to-peer (P2P) networks. The web content harvested by a distributed crawler can be indexed by decentralized search infrastructures, or archived using a persistent storage infrastructure. The distributed query processor naturally coordinates and partitions the work across the participating nodes. Using only data partitioning of the intermediate URLs to be processed, it parallelizes the crawl without explicit code to ensure that multiple sites do not crawl the same web pages redundantly.
A distributed crawler (see Figure 2.11) consists of three subsystems [61, 66] Coordinator Subsystem, Mobile Agents Subsystem, and a Public Search Engine. The Coordinator subsystem resides at the search engine site and is responsible for administering; the Mobile Agents subsystem is responsible for crawling. It is responsible for maintaining the database with the crawling results that it gets from the migrating crawlers.

**Figure 2.11:** General Architecture of a Distributed Web Crawler

The Mobile Agents subsystem [66] is divided into two categories Migrating Crawlers and Data Carries. Migrating crawlers are responsible for on-site crawling and monitoring of remote web servers. They process the crawled pages, and send the results back to the coordinator subsystem. Data Carries are responsible for transferring the processed and compressed information from the Migrating crawlers back to the Coordinator subsystem. The Public search engine executes user queries on the database maintained by the Coordinator subsystem.

Parcahyd [63] searches the desired information in parallel thereby minimizing the time of retrieval. Parallelization of crawling system is necessary for downloading documents in a reasonable amount of time. The work has done reported here to focuses on providing parallelization at three levels: the document, the mapper, and the crawl worker level. In this architecture bottleneck at the document level has been removed.
2.9.2 FOCUSED CRAWLERS

A focussed crawler [61, 69, 70] only gathers documents on a specific topic, thus reducing the amount of network traffic and downloads. A focussed crawler attempts to download only web pages that are relevant to pre-defined topic(s). The goal of a focussed crawler is to select links that lead to documents of interest, while avoiding links that lead to irrelevant topics. Unlike an exhaustive crawler which follows each link on a page in breadth first manner, the focussed crawler gives priority to links that belong to pages classified as relevant. The crawler uses an additional classifier to select the most promising links on a relevant page.

The focussed crawler seeks, acquires, indexes, and maintains pages on a specific set of topics that represent a relatively small portion of the web. It crawler starts with a seed list which contains URLs that are relevant to the topic of interest, it crawls these URLs and then follows the links from these pages to identify the most promising links based on both the contents of the source pages and the link structure of the web. Three main components of a focussed crawler are Classifier, Distiller and Crawler as shown in Figure 2.12. Classifier makes relevance judgments on pages crawled to decide on link expansion. Distiller determines a measure of centrality of crawled pages to determine visit priorities. Crawler with dynamically reconfigurable priority controls is governed by the classifier and distiller.

![General Architecture of a Focussed Web Crawler](image)

**Figure 2.12:** General Architecture of a Focussed Web Crawler
The most crucial evaluation of focused crawling is to measure the *harvest ratio*, which is the rate at which relevant pages are acquired and irrelevant pages are effectively filtered off from the crawl. This harvest ratio must be high, otherwise the focused crawler would spend a lot of time merely eliminating irrelevant pages, and it may be better to use an ordinary crawler instead.

### 2.9.3 FORM FOCUSED CRAWLERS

A Form focussed crawler [61] deals with the sparse distribution of forms on the web (see Figure 2.13). 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.

![General Architecture of a Form focussed Web Crawler](image.png)

**Figure 2.13:** General Architecture of a Form focussed Web Crawler

### 2.9.4 HIDDEN WEB CRAWLERS

The Hidden web [61] consists of files, images and websites that cannot be indexed by traditional web crawlers. Traditional crawlers rely on the hyperlinks on the web to discover pages. Due to lack of links, these search engines can’t index the hidden web pages. Specifically a hidden web crawler performs a sequence of actions for each form on a page i.e. *Form Analysis, Value assignment and submission, Response Analysis and Response Navigation.*
The architecture of a hidden web crawler includes *six basic functional modules and two internal crawler data structures* as shown in Figure 2.14. The basic data structure is the *URL List* that contains all the URLs that the crawler has discovered so far. The *Crawl Manager* controls the entire crawling process, it decides which link to visit next, makes the network connection to retrieve page from the web, and handover the downloaded page to the *Parser* module. The *Parser* extracts hypertext links from the page and adds them to the URL List. This sequence of operations is repeated until some termination condition is satisfied.

![Figure 2.14: General Architecture of a Hidden Web Crawler](image)

### 2.9.5 INCREMENTAL WEB CRAWLERS

A crawler needs to maintain freshness of its local collection. To maintain up-to-date-ness it revisits the web pages, and depending upon the way it revisits crawlers can be classified as Batch-mode crawlers or Steady crawlers.

A *batch-mode crawler* [8] runs periodically, say once in a month and updates all pages in the local collection in each crawl (see Figure 2.15(a)). In the graph drawn between freshness of pages and time (see Figure 2.15(b)), horizontal axis represents time and
vertical axis represents freshness. The curve in the graph shows freshness changes over time, and the dotted line shows averaged freshness over time. It is also assumed that the pages crawled are updated in the local collection.

It is evident from the Figure 2.15(b) that the collection gets fresher in grey region (i.e. when the crawler revisits web pages), and the collection gets decayed when the crawler is idle (i.e. freshness decreases in white region). Moreover, the freshness is not equal to one even when a crawl completes, as few pages have already been changed during the crawl. Also, the freshness of the local collection declines exponentially in white region i.e. when crawler is not running.

![Operation of a Batch-mode Crawler](image)

![Freshness in a Batch-mode Crawler](image)

**Figure 2.15:** Operation and Freshness in a Batch-mode Crawler
In contrast, a steady crawler [8] runs non-stop (see Figure 2.16(a)) without any pause. The freshness of a steady crawler (see Figure 2.16(b)) is stable over time, as the collection is continuously and incrementally updated. On analyzing freshness of a batch mode crawler (see Figure 2.15(b)) and a steady crawler (see Figure 2.16(b)), it may be observed that the average freshness of both is same. However, to achieve same freshness level, a batch mode crawler needs to visit pages at a high speed as compared to a steady crawler that can do the same at a lower speed working continuously. It is good to run a crawler at average peak speed continuously rather than periodically with high speed that increases unnecessary congestion on the busy network.

![Diagram](image)

(a) Operation of a Steady Crawler

![Diagram](image)

(b) Freshness in a Steady Crawler

**Figure 2.16:** Operation and Freshness in a Steady Crawler

For achieving the freshness, a crawler may update/replace the old version of a page with new in two ways i.e. in-place updation or shadowing [8]. In *in-place updation* the new
page replaces the old page immediately which allows a user to get fresh information all the time. Whereas in shadowing a new set of pages is collected from the web and stored in a separate space from the current location and, when crawling is completed and all the pages are collected and processed, the old/current collection is replaced with new collection.

When batch mode crawling is done with shadowing updation and fixed frequency of visit to pages, it is called a periodic web crawler [8]. It is easy to implement and all operations (storage, indexing) are held with the new collection and, the old/current collection remains intact that gives higher availability of old/current collection to the users. When steady crawling is done with in-place updation with a variable frequency of visits to web pages, it is called an incremental web crawler.

The periodic crawler visits the websites until its collection has a desirable number of pages and stop visiting pages. Whenever it needs to refresh its collection, it revisits the sites, creates a new collection and replaces the old collection with the new. Whereas an incremental crawler refresh existing pages and replaces less important existing pages with more important new pages. It crawls the websites continuously, refreshes local collection and provides fresh information to the user. Good freshness can only be guaranteed significantly by simply revisiting all pages very frequently and selecting the page that will increase the freshness most significantly.

The design of an incremental crawler needs to address the following issues [8]:-

- **Keep the local collection fresh**: The freshness of pages in local collection depends on the strategy used, so the crawler should use the best policies to keep the local collection fresh. This includes adjusting the revisit frequency for a page based on its estimated change frequency.

- **Improve quality of the local collection**: The crawler should increase quality of the local collection by replacing less important pages with more important pages. It is necessary because of two reasons; firstly, pages are constantly created and removed, and some of the pages created may be more important than existing pages in the local collection. So, the crawler needs to replace less important existing pages with more important new pages. Secondly, the importance of
existing pages also changes over time. So, when some existing pages become less important than previously ignored pages, the crawler should replace less important existing pages with previously ignored new pages.

It maintains three main data structures ALL_URLs, COLL_URLs and LocalCollection, and three main software modules RankingModule, UpdateModule and CrawlModule as shown in Figure 2.17.

![General Architecture of an Incremental Web Crawler](image)

**Figure 2.17:** General Architecture of an Incremental Web Crawler

It consists of three major modules i.e. RankingModule, UpdateModule and CrawlModule and three data structures i.e. ALL_URLs, COLL_URLs and LocalCollection. ALL_URLs records all URLs that the crawler has discovered, and COLL_URLs records the URLs that are/will be in the LocalCollection.

### 2.10 DOCUMENT REFRESH TECHNIQUES

Due to the lack of efficient refresh techniques [71], current crawlers add unnecessary traffic to the already overloaded Internet. Frequency of visits to sites can be optimized by calculating refresh time dynamically. It helps in improving the effectiveness of the crawling system by efficiently managing the revisiting frequency of a website; and appropriate chance to each type of website to be crawled at appropriate rate.
Using migrants (i.e. migrating crawlers), the process of selection and filtration of web documents can be done at web servers rather than search engine side which can reduce network load caused by the web crawlers. Frequency of visits to sites can be optimized by dynamically assigning a priority to a site. The computation of refresh time helps in improving the effectiveness of the crawling system by efficiently managing the revisiting frequency of a website; and appropriate chance to each type of website to be crawled at a fast rate.

The process of revisiting to a website can further be improved by adjusting the frequency of visit by considering the interest of users shown for specifics websites. For example, the websites for which users show more interest be crawled at a faster rate as compared to those that are less or rarely surfed by the users.

2.10.1 VARIOUS APPROACHES FOR REFRESH TECHNIQUES

The agent approach use the bandwidth of the network to migrate an agent to a platform, and allow it to continue to run after leaving a node, even if they lose connection with the node where they were created thereby provide the better utilisation on communication and allows parallel distributed applications. An agent can move on to other machines when necessary and can delegate tasks to other mobile agents in order to achieve real parallel applications. Various studies have shown that distributed crawling methods based on migrating crawlers are essential tools for allowing such access that minimizes network utilization and also keeps up with document changes.

Due to the deficiency in their refresh techniques [9, 72], current crawlers add unnecessary traffic to the already overloaded Internet. Moreover there exist no certain ways to verify whether a document has been updated or not. Studies held on revisitation have demonstrated that 50% to 80% [6, 73, 74] of all web surfing behavior involves pages that users have previously visited. While many revisits occur shortly after a page’s first visit (e.g., during the same session using the back button), a significant number occur after a considerable amount of time has elapsed [75].

Bullot et al [76] introduced data mining approach for optimizing performance of an Incremental Crawler. With the method presented, it is the user who chooses which pages the crawler must update. Kuppusamy and Aghila [77] involves user participation in larger extent in order to get the focused and more relevant information. Choudhari and Choudhari [78] address the scheduling problem and solution for the web crawlers with
the objective of the optimizing the resources like freshness of repository and the quality of the index. They divided the web content providers into two parts i.e. active and inactive. For inactive content providers they use agents who continuously crawls the content providers and collect the update pattern of the content providers.

Glover et al [79] described a meta search engine architecture, that allows users to provide preferences in the form of an information need category. This extra information is used to direct the search process, providing more valuable results than by considering only the query. Using this architecture, identical keyword queries may be sent to different search engines, and results may be scored differently for different users. Malakar [80] presents a novel approach to personalised search using the concept of “iAGENT” an intelligent agent that assists a user to get relevant documents by modifying the query given by the user in accordance with the web pages previously visited. It presents a novel approach to personalise the search results and improve the relevancy rate.

Qiu and Cho [81] shows how a search engine can learn a user’s preference automatically based on her past click history and how it can use the user preference to personalize search results. They propose a framework to investigate the problem of personalizing web search based on users’ past search histories without user efforts. Based on this correlation, they describe an intuitive algorithm to actually learn users’ interests. They propose two different methods, based on different assumptions on user behaviors, to rank search results based on the user’s interests we have learned.

Liu et al [82] proposes to measure page importance through mining user interest and behaviors from web browse logs. Unlike most existing approaches which work on single URL, here, both the log mining and the crawl ordering are performed at the granularity of URL pattern. The proposed URL pattern-based crawl orderings are capable to properly predict the importance of newly created (unseen) URLs. Vipul et al approach [83] is build on the basis of which a web crawler maintains the retrieved pages “fresh” in the local collection. Towards this goal the concept of Page Rank and Age of a web page is used. As higher page rank means that more number of users are visiting that very web page and that page has higher link popularity. Age of web page is a measure that indicates how outdated the local copy is. Using these two parameters a hybrid approach is proposed that can identify important pages at the early stage of a crawl, and the crawler re-visit these important pages with higher priority.
Dixit et al [11, 12] propose an efficient approach for building an effective incremental web crawler with an approach for optimizing the frequency of visits to sites. The approach adjusts the frequency of visit by dynamically assigning a priority to a site. A mechanism for computing the dynamic priority for any site has also been developed.

An alternate approach [7] to manage the process of revisiting of a website, employs an ecology of crawl workers to crawl the websites. Crawl manager extracts URLs from each queue of URLs and distribute them among crawl workers. The architecture manages the process of revisiting of a website with a view to maintain fairly fresh documents at the search engine site. The computation of refresh time helps in improving the effectiveness of the crawling system by efficiently managing the revisiting frequency of a website; and appropriate chance to each type of website to be crawled at a fast rate.

2.11 NEED OF STORING INDEX

In a search engine process, the downloaded pages are indexed and stored in a database. The indexer extracts all the uncommon words (keywords) from each page and records the URLs where each word has occurred. The result is stored in a large table containing URLs; pointing to pages in the repository where a given keyword occurs. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, normally incorporated as an inverted index, the search engine would scan every possible document on the web, which would require considerable time and computing power; impossible with the current web size. Keyword searching is the most common form of text search on the web. Inverted indices are one of the most commonly used techniques to organise very large document collections and provide high-speed access to set of documents satisfying queries.

This index can only determine whether a word exists within a particular document, since it stores no information regarding the frequency and position of the word. It is therefore considered to be a Boolean index. Such an index identifies documents matching a query but does not rank matched documents. In some designs index includes additional information such as the frequency of each word in each document or the positions of a word in each document. Position information enables the search algorithm to identify word proximity to support searching for phrases; frequency can be used to help in ranking the relevance of documents to the query [84].
2.11.1 INVERTED INDEXING

Inverted indices are one of the most commonly used techniques to organise very large document collections. In view of high speed searching, the inverted index structure [85, 86] is widely used in the modern super fast search engines like Google, Yahoo, Lucene and other major search engines. They provide high-speed access to sets of documents satisfying queries, which can be subsequently ranked and returned to the user. If designed properly these are also extremely compact [87]. The inverted index is a technique based on words to make an index for the text to enhance the speed of search activities [88]. The classic inverted index form consists of a text word and its occurrence which enumerates its positions within each document.

The inverted index maps back from terms to the parts of a document where they occur. Inverted index (see Figure 2.18) keeps a dictionary of terms (vocabulary or lexicon). For each term, it has a list that records which documents the term occurs in. Each item in the list which records that a term appeared in a document (and, later, often, the positions in the document) is conventionally called a posting. The list is then called a postings list (or inverted list), and all the postings lists taken together are referred to as the postings. The dictionary has been sorted alphabetically and each postings list is sorted by document ID.

![Figure 2.18: An Inverted Index](image)

To gain the speed benefits of indexing at retrieval time, it needs to build the index in advance. The major steps in this process are; collecting the documents to be indexed, tokenizing the text turning each document into a list of tokens, do linguistic pre-processing producing a list of normalized tokens which are the indexing terms, and index
the documents that each term occurs in by creating an inverted index consisting of a dictionary and postings.

Within a document collection, it is assumed that each document has a unique serial number, known as the document identifier (docID). The core indexing step is, sorting this list so that the terms are alphabetical. Multiple occurrences of the same term from the same document are then merged. Instances of the same term are then grouped, and the result is split into a dictionary and postings. Since a term generally occurs in a number of documents, this data organization already reduces the storage requirements of the index. The dictionary also records statistics, such as the number of documents containing each term.

When the index is ready, the searching can be performed through query interface, a user enters a query into a search engine by using keywords, the engine examines its index and provides a listing of best matching web pages according to its criteria, usually with a short summary containing the document's title and sometimes parts of the text. In this stage the result is ranked, where Ranking is a relationship between a set of items such that, for any two items the first is either “ranked higher than”, “ranked lower than” or “ranked equal” to the second. Ranking is done according to document relevancy to the query, freshness and popularity [86]. Most search engines and information retrieval systems use inverted indexes as their main data structure for full-text indexing [88].

Jasen et al [89] indicated that approximately 80% of the users examine at most the first three batches of the results i.e. 80% of the users typically view at most 30 to 60 results for every query that they issue to a search engine. Since the users are interested in a small number of results, using an index that is capable of returning all the results for a query may constitute a significant waste in terms of time, storage space and computational resources, which is bound to get worse as the web grows larger over time.

Farragina et al [88] proposed two new compressed representations for general sequences, which produce an index that improves over the one in by removing from the query times the dependence on the alphabet size and the poly-logarithmic terms. They further presented [90] the existing implementations of compressed indexes which offer tuned implementations and a standardized API for the most successful compressed full-text self-indexes. Ntoulas et al [91] in their paper show how to determine the optimal size of a
pruned index and experimentally evaluate algorithms on a collection of 130 million web pages. Hao et al [86] focus on optimizing compression methods and query processing. They also propose and evaluate techniques for compressing frequency values for this case.

### 2.11.2 SEARCHING FOR SYNONYMS

A number of techniques have been proposed for searching for synonyms of the keywords user is intended to search for. Sanchez et al [92], states that the search of web resources is a very important topic due to the huge amount of valuable information available in the web. Standard search engines can be a great help but they are often based only on the presence or absence of keywords. In this, problems regarding semantic ambiguity appear. They propose a new method for discovering lexicalizations and synonyms of search queries based on a previously obtained taxonomy of terms for the specified domain.

Meusel [93] presents that maintaining and extending large thesauri is an important challenge facing digital libraries and IT businesses alike. In their paper they describe a method building on and extending existing methods from the areas of thesaurus maintenance, natural language processing, and machine learning to (a) extract a set of novel candidate concepts from text corpora and (b) to generate a small ranked list of suggestions for the position of these concept in an existing thesaurus. Based on a modification of the standard $tf-idf$ term weighting they extract relevant concept candidates from a document corpus, then apply a pattern-based machine learning approach on content extracted from web search engine snippets to determine the type of relation between the candidate terms and existing thesaurus concepts.

S. Agrawal et al [94] introduced the ad-hoc entity extraction task where entities of interest are constrained to be from a list of entities that is specific to the task, and have also considered approximate match based on similarity functions. Surajit Chaudhuri et al [95] discuss the tasks of recognizing named entities such as products, people names, or locations from documents. In their paper, they exploit web search engines in order to define new similarity functions.

Xing Wei et al [96] presents that search with synonyms is a challenging problem for web search, as it can easily cause intent drifting. In their paper, they propose a practical
solution to this issue, based on co-clicked query analysis, i.e., analyzing queries leading to clicking the same documents. They have developed a synonym discovery approach based on co-clicked query data, and improved search relevance and user experience significantly, based on the approach. Approximate-match based dictionary lookup was studied under the context of string similarity search in application scenarios such as data cleaning and entity extraction [95]. All these techniques rely on similarity functions which only use information from the input string and the target entity it is supposed to match. All the papers discussed here, present use of inverted indexing for index preparation, and shortening or optimizing it.

2.12 PROBLEMS WITH TRADITIONAL APPROACH

The main problems with the way current search engines index the web, using traditional approach are [97, 98]:-

- **Scaling:** The concept of “download-first-and-index-later” does not scale given the limitations in the infrastructure and projected growth rate of the web. Using the estimates for growth of web indices, a web crawler running in the year 2000 would have to retrieve web data at a rate of 45Mbit per second in order to download the estimated 480GB of pages per day that are necessary to maintain the index. Looking at the fundamental limitations of storage technology and communication networks, it is highly unlikely that web indices of this size can be maintained efficiently.

- **Efficiency:** Existing search engines add unnecessary traffic to the already overloaded Internet. While current approaches are the only alternative for general-purpose search engines trying to build a comprehensive web index, there are many scenarios where it is more efficient to download and index only selected pages. These are called specialized search engines.

- **Quality of Index:** The results of web searches are overwhelming and require the user to act as part of the query processor. Current commercial search engines maintain web indices of up to 110 million pages and easily find several thousands of matches for an average query. Thus increasing the size of the web index does not automatically improve the quality of the search results if it simply causes the search engine to return twice as many matches to a query as before.
Search engines download parts of the existing web and offer the Internet users access to this database through keyword search. To create a search engine which scales today's web is a non-trivial task. It requires fast crawling technology to gather the web documents and keep them up-to-date. Storage space is required to store web documents and their indices. An indexing system is required to process billions of records efficiently. Moreover, queries are to be handled at a very fast rate, say thousands per second.

A crawler has to be polite and efficient, trying not to harm or overload the web servers it is visiting. They need to function somewhere between the cushion of Moore's Law and the hard place of the exponential growth of the web [99]. With such exponential growth currently being maintained, the trends towards automated production of web pages from databases make it likely that such growth will continue, or even accelerate, in the future. With the available bandwidth for conducting crawls which is neither infinite nor free, it is becoming essential to crawl the web in not only scalable but efficient way if some reasonable measure of quality or freshness is to be maintained. Also, it is desirable to minimize the number of pages with low PageRank [13,100] in the local collection, and one may achieve such objective in a number of ways. Several studies of web crawling have been made, some have concentrated on aspects relating to caching, others have been principally interested in the most efficient and effective way to update axed size database extracted from the web, often for some specific function, such as data mining. Widely accepted Poisson model forms the basis for a series of studies on crawler strategies.

It is evident that 40% of current Internet traffic and bandwidth consumption is due to the web crawlers that retrieve pages for indexing by the different search engines, and as the size of the web continues to grow, this network load will also increase further. The centralized crawling techniques are unable to cope up with constantly growing web. Distributed crawling methods based on migrating (or mobile) agents, are an essential tool for allowing such access that minimizes network utilization and also keeps up with document changes. The mobile code from search engine side transfers and executes on web servers, an environment controlled by another party, it gives rise to several security issues in mobile agent computing like integrity of mobile agents and their authentication, authorization, intrusion detection etc.

The next chapter presents a view on software agents and their types. It also discusses mobile agents, their design issues and applications. Thereafter it presents the need of
maintaining integrity and security of free roaming agents, the remote platform where they execute and data they carry, and various issues and approaches related to security.