CHAPTER 4

DOMAIN SPECIFIC AND INCREMENTAL CRAWLING

The size of the internet is large and it had grown enormously search engines are the tools for Web site navigation and search. Search engines maintain indices for web documents and provide search facilities by continuously downloading Web pages for processing. This process of downloading web pages is known as web crawling. In this chapter the architecture for Effective Migrating Parallel Web Crawling approach with domain specific and incremental crawling strategy is proposed that makes web crawling system more effective and efficient. The major advantages of migrating parallel web crawler are that the analysis portion of the crawling process is done locally at the residence of data rather than inside the Web search engine repository. This significantly reduces network load and traffic which in turn improves the performance, effectiveness and efficiency of the crawling process. The characteristic of migrating parallel crawler is that as the size of the Web grows, it becomes necessary to parallelize a crawling process, in order to finish downloading web pages in a comparatively shorter time. Domain specific crawling will yield high quality pages. The crawling process will migrate to host or server with specific domain and start downloading pages within specific domain. Incremental crawling will keep the pages in local database fresh thus increasing the quality of downloaded pages.

4.1 Introduction

In this chapter an effective domain specific and incremental crawler is designed and implemented. The crawler in the domain specific manner selectively and incrementally updates its index and local collection of pages. The incremental crawler can improve the freshness of the collected documents and bring in new pages in the collection. The
proposed architecture for domain specific and incremental crawler combines the best design choices. Crawler starts with a set of URLs called seed URLs. Firstly it retrieves the pages specified by the seed URLs. Then it extracts URLs in those pages. The new URLs are added to queue of URLs to be downloaded. This process is repeated for the URLs from the queue. The web crawler can update its collection of documents in following ways: the web crawler visits the World Wide Web until the collection of documents reached the desired number of pages and then the crawler stops visiting pages. The web crawler then refreshes the collection of documents. The web crawler replaces the old collection of documents with new one. The crawler following this technique is called periodic crawler. The web crawler keeps on visiting pages after the collection reaches its maximum size to incrementally update and refresh the collection of documents. By this method the web crawler refreshes existing pages and also replaces pages that are less important with the pages that are new and are of more-importance. The crawler following this technique is called incremental crawler. The incremental crawler is more effective than the periodic crawler. If the Web crawler is able to estimates the rate of change of web pages then the incremental web crawler need to crawl only the pages that have changed, instead of crawling the whole collection of documents. This technique saves network bandwidth and improves the freshness of the collected documents. The incremental web crawler collects new web pages in a timelier manner than the periodic web crawler. The periodic web crawler collects a new page when the next crawling cycle starts while incremental web crawler may revisit the page immediately. Web search engines are important searching tool for the information available on the Internet. Research in these areas can improve users experience on the Internet. If web pages change at same time interval then there would have been no differences between incremental and periodic crawler. These two crawling techniques would have revisited pages after the same time interval. The web is dynamic in nature large numbers of pages are added, updated every month. In this chapter domain specific and incremental crawler is designed and implemented. The evolution of web is studied and design choices for web crawler are compared. The advantages and disadvantages of periodic and an incremental crawler are discussed.
4.2 Problems in Generic Web Crawlers

Web crawlers of big commercial search engines crawl up to 10 million pages per day. Assuming an average page size of 6K, the crawling activities of a single commercial search engine adds a daily load of 60GB to the Web. The issues in generic web crawlers are scaling issues, efficiency issues and index quality issues.

Scaling Issues: One of the first Web search engines, the World Wide Web Worm, was introduced in 1994 and used an index of 110,000 Web pages. Big commercial search engines in 1998 claimed to index up to 110 million pages. 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.

Efficiency Issues: Current Web crawlers download all these irrelevant pages because traditional crawling techniques cannot analyze the page content prior to page download.

Index Quality Issues: Current commercial search engines maintain Web indices of up to several hundred million pages. That is why the quality of index is compromised.

4.3 Issues in Migrating Parallel Web Crawlers

The Migrating Parallel Crawler system consists of Central Crawler, Crawl Frontiers, and Local Database of each Crawl Frontier and Centralized Database. It is responsibility of central crawler to receive the URL input from the applications and forwards the URLs to the available migrating crawling process. Crawling process migrate to different machines to increase the system performance. Local database of each crawl frontier are buffers that locally collect the data. This data is transferred to the central crawler after compression and filtering which reduces the network bandwidth overhead. The central crawler has centralized databases which contain the documents collected by the crawl frontiers independently. The main advantages of the migrating parallel crawler approach are Localized Data Access, Remote Page Selection, Remote Page Filtering, Remote Page Compression, Scalability, Network-load dispersion, Network-load reduction.

The following issues are important in the study of a migrating parallel crawler:
Communication bandwidth: Communication is important as it prevents overlap and it improves the quality of the downloaded content. Crawling processes need to communicate and coordinate with each other to improve quality while consuming less communication bandwidth.

Quality: Prime objective of migrating parallel web crawler is to download the “important” pages first in order to improve the “quality” of the collected pages.

Overlap: When multiple processes run in parallel to download pages, it is possible that different processes download the same page multiple times. One process may download the same page that other process may have already downloaded, one process may not be aware that another process has downloaded the same page.

The main aim of incremental crawler is to maintain the local collection up-to-date or to maintain the collection fresh. Freshness represents the pages that are up-to-date in the local database. When the web pages in the collection are same as the current state on the Internet then the freshness of the collection is said to be 1, while when half of the collection is up-to-date then the freshness is said to be 0.5. A Web crawler performs the required revisit to web pages in order to maintain the local database fresh and up-to-date. The web crawler can be categorized as Batch-mode crawler and Steady crawler.

Batch-mode crawler: A batch-mode crawler runs periodically (eg. bimonthly) updating all pages in the collection in each crawl. When the web crawler is idle the freshness of the collection decreases, when the web crawler revisits pages the collection of documents gets fresher. The freshness of the collection could not be 1 at the end of each crawling cycle since some pages have already changed.

Steady crawler: A steady crawler runs continuously without any pause. In case of steady crawler the freshness is stable because the collection is continuously and incrementally updated. However freshness for both the batch mode web crawler and steady web crawler evolve differently but if they visit pages at the same average frequency their freshness is same when averaged over time. The steady crawler collect pages at a lower peak speed. The batch mode web crawler must visit pages at a higher speed. This increases the peak load on the web crawler local machine. The peak crawling speed is a
very sensitive issue on the web. The webmasters of web sites trace web crawler that accesses their sites. If web crawler runs too fast then it is completely blocked from accessing web sites.

4.3.1 Collection Updating in Place

When a web crawler replaces an old web page with a new version of the same web page then it is called shadowing. In shadowing technique a new set of web pages are collected and stored in separate new location. After all new web pages are collected the current collection is replaced with new collection. The collection stored in the shadowing space is called crawler’s collection and the collection that is available to users is called as current collection. Shadowing improves the availability of the current collection as it is shielded from the crawling process. The current collection provides response to requests of user while at the same time the crawler’s collection is pre-processed so that it can be made available to the end users. Shadowing is easier to implement than in place updates because updating and indexing processes are separate from the access processes. Shadowing decreases freshness.

The crawler current collection is replaced by the crawler’s collection after all pages are collected. For steady crawler new pages are collected from scratch say every month, this improves the freshness of the crawler’s collection. At the end of each month current collection is replaced by the crawler’s collection. After that freshness of the current collection decreases, until the current collection is replaced by a new set of pages. Freshness of the current collection is higher without shadowing. For the batch-mode crawler freshness is same for shadowed collection or un-shadowed collection. When the crawler is running, the freshness of the in-place update crawler is higher than shadowing crawler as new pages are available to users with the in-place update crawler. Freshness of the steady crawler decreases with shadowing while the freshness of the batch-mode crawler is not affected by shadowing. For a steady crawler, in-place updates may be a good strategy.
4.3.2 Page Refresh Frequency

As web crawler updates pages it visit the web pages with same frequency or with variable frequencies.

Fixed frequency: The web crawler revisits pages at the same frequency irrespective of change in web documents. Fixed-frequency strategy is used by a batch-mode crawler, since a batch-mode crawler revisits all pages in the collection.

Variable frequency: The web pages change at different rates. Based on the page change the web crawler optimizes the revisit frequency for a page. The variable frequency strategy is used in steady crawler with in-place updates. Steady web crawler can adjust the revisit frequency thus increasing the freshness of web documents. For variable frequency the web crawler needs to decide the rate at which the web page is to be visited. The web crawler should revisit a page when it changes. If the page is changed more often, then the web crawler should visit the page more often. But this is not always right as illustrated by the example.

Suppose that there are two pages p₁ and p₂. Page p₁ changes every second and page p₂ changes every day. The web crawler can crawl only one page per day due to bandwidth limitations. If the web crawler revisits page p₁, p₁ will remain up-to-date for a half second so the freshness will be 0.5 only for a half second. If the web crawler revisits page p₂, p₂ will remain up-to-date for a half of the day. Therefore, the freshness of the collection will be 0.5 for a half of the day. Therefore it is better to visit p₂ (which changes less often than p₁), than to visit p₁. Hence optimal revisit frequency is not always proportional to the change frequency of a page. The incremental crawler gives us high freshness and results in low peak loads. The periodic crawler is easier to implement and interferes less with a highly utilized current collection.

4.4 Coordination in Migrating Parallel Web Crawler

The coordination is done in order to prevent overlap and also crawling processes need to coordinate with each other so that the correct and quality pages are downloaded. This coordination can be achieved in following ways:
Independent: In this method Crawling processes may download pages independent of each other without any coordination. The downloaded pages may overlap in this case, but the assumption is that overlap is not significant enough.

Static assignment: In this method the Web is partitioned into sub partitions and each sub partition is assigned to each Crawling processes before the actual crawling process may begin. This type of arrangement is called static assignment.

Dynamic assignment: In this coordination scheme there is a central coordinator that divides the Web into small partitions and each partition is dynamically assigned to a Crawling processes for further downloading.

4.5 Crawling Modes for Static Assignment

In this scheme of static assignment, each Crawling processes are responsible for a certain partition of the Web and have to download pages within the partition. There may be possibility that some pages in the partition may have links to pages in another partition. Such types of links are referred to as an inter-partition link. The following are crawling modes for the static assignment.

Firewall mode: Each Crawling processes strictly download the pages within its partition and inter-partition links are not followed. In firewall mode all inter-partition links are either thrown away or ignored.

Cross-over mode: In this mode each Crawling processes download pages within its partition. It follows inter-partition links when links within its partitions are exhausted.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Coverage</th>
<th>Overlap</th>
<th>Quality</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firewall</td>
<td>Bad</td>
<td>Good</td>
<td>Bad</td>
<td>Good</td>
</tr>
<tr>
<td>Cross-Over</td>
<td>Good</td>
<td>Bad</td>
<td>Bad</td>
<td>Good</td>
</tr>
<tr>
<td>Exchange</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Bad</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of three Crawling Modes [16]
Exchange mode: In this mode, crawling processes periodically exchange inter-partition links. Crawling processes are not allowed to follow inter-partition links [16]. Table 4.1 compares the three crawling modes.

**4.6 Evaluation Model in Migrating Parallel Web Crawler**

This section deals with metrics that are defined to quantify the advantages or disadvantages of migrating parallel crawler.

Coverage: the coverage is defined as $I/U$, where $U$ is the number of pages downloaded by migrating parallel crawler and $I$ is the number of unique pages downloaded by the migrating parallel crawler.

Overlap: The overlap is defined as $(N-I)/I$. Where, $N$ is the number of pages downloaded by the crawler and $I$ is the number of uniquely downloaded pages by migrating parallel crawler.

Quality: The quality of downloaded pages is defined as $|A_N \cap P_N|/|P_N|$. The migrating parallel crawler downloads the important $N$ pages, and $P_N$ is that set of $N$ pages. $A_N$ is set of $N$ pages that an actual migrating parallel crawler would download, which is different from $P_N$.

Communication overhead: The communication overhead is defined as $E/I$. Where $I$ is total pages downloaded and $E$ represents exchanged inter-partition URLs.

Table 4.2 summarizes different crawling techniques with their performance attributes. The “++” indicate that particular performance attributes is present in the Crawler and “--” indicates that the particular performance attributes is not present in the Crawler.

Crawling process of migrating parallel crawlers move to the resource which needs to be crawled to take the advantage of localized data access. After accessing a resource, migrating parallel web crawler move on to the next machine or server and send result of crawling to central database in compressed form. From the table it is evident that Migrating parallel web crawlers are more efficient in terms of network load reduction, I/O performance, reducing communication bandwidth, network load dispersion etc.
<table>
<thead>
<tr>
<th></th>
<th>High performance distributed crawler</th>
<th>Incremental crawler</th>
<th>Parallel crawler</th>
<th>Agent based</th>
<th>Migrating parallel Crawler</th>
<th>Domain Specific Crawler</th>
<th>Mobile Crawler</th>
<th>Breadth First Crawling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robustness</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Flexibility</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Manageability</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>I/O Performance</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Network resources</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>OS limits</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Higher performance</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Extensibility</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Incremental crawling</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Reasonable cost</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Overlapping</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Communication bandwidth</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Network load dispersion</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Network load reduction</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Freshness</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Page rank</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
</tr>
<tr>
<td>Scalability</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Load sharing</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>High quality</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Agent oriented</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 4.2: Performance Attributes for Different Web Crawlers
4.7 Incremental Crawling

Different types of crawling strategies have emerged they are broad, focused, continuous and incremental crawling. Broad and focused crawling strategies are very similar except that broad crawling strategy captures a large scope while focused crawling strategy captures complete coverage of a smaller scope. Both these approaches crawls the scope once only. This is called snap shot strategy. Crawls are repeatable if the crawling again starts from the seed URL. The snapshot strategy is also called periodic crawling. Periodic crawling is useful in large scale crawling. Large crawling takes a lot of time, which implies that there are time intervals between revisits. Snapshot crawling strategy fails to detect the unchanged documents, which leads to duplication of web pages. Snapshot crawling strategy is used for large scale crawling i.e., crawling a large number of websites or to crawl each website completely.

In continuous crawling strategy the crawler visit the same resources at certain time intervals. In this strategy the crawler must contain detailed information of state. In continuous crawling an incremental strategy is used instead of snapshot strategy. A record is maintained for history of resource in incremental strategy which is used to determine priority of resource to be fetched. Adaptive revisiting techniques are used to capture the changes in online resources thus allowing the crawler to revisit each page more frequently. In snap shot approach revisits are bounded by crawling cycles while incremental strategy is not bounded by crawling cycles. Since any page can be revisited any time, rather than only once during the crawling process, snap shot strategy can cover a broad scope while in an incremental strategy the scope is limited because of the need to revisit resources and overheads. Both the incremental and snapshot strategy can be considered as space and time completeness. Due to limited bandwidth and storage the balance has to be maintained between the two strategies. Proposed work implements a combination of snapshot strategy for focus crawling and adopts the incremental strategy for expanding its capabilities for broad crawls.

Incremental strategy captures change in all the resources within its scope. This technique can be applied where the scope is limited. Incremental strategy requires a series of request to detect changes. However in this strategy server will be heavily loaded with requests. The best technique is to crawl the entire domain then restart the
crawl again. The resources that are discovered during iterations are preserved while resources that are not appearing in successive crawls are removed from consideration. It is known fact that the resources tend to change within the same timeframe. A method is required that predict how often a resource tends to change in a given time interval. In other words after what time interval the crawler should revisit the resource such that all the versions of the resource are captured.

The method computes the probability of change and computation is used by the crawler to revisit resource after the appropriate time interval. In order to understand the method let us consider some known facts about the resources on the Internet and documents on the web. A resource that tends to change often is most likely to change in future also. The document changes more frequently should be revisited more frequently. The document that does not change frequently is not required to be revisited frequently. This cannot be considered as universal law but it tends to happen with certain exceptions. Mostly web resources are believed to have reasonably good regular change and update rates. Generally the web sites that have regular readers are required to be consistent in changing their contents. The type of file that is being used as resource determines the probability of change. HTML or text files are easy to change while images are difficult to change. HTML files are generated on demand while images and multimedia files are not generated on demand. The graphics file is usually not replaced, it is updated. Thus waiting time can be predicted between two successive crawls. HTML and text files can be assigned a short waiting time between two successive crawls. A longer initial waiting time between two successive crawl is assigned to image and multimedia files. PDF documents and formatted Word documents are also less likely to change. Audio and video files have even low probability to change. Their waiting time between two successive crawls, can be higher as compared to HTML files. There is relationship between the size of a resource and its change rate. As the size of files increases it change rate decreases. Thus image, audio, video and formatted files are larger than pure text documents. This information is used to reduce the number of visits between two successive crawls. Location of document also reflects certain detail about change in web documents. The front page of a web site tends to changes more than any other page. Seed URLs are the front pages of web site. Seed URLs are used as a factor in deciding the waiting time between successive crawls. The incremental crawling strategy
is used to utilize bandwidth more effectively and in reducing storage costs, since duplicate data need not be stored. However the drawback in this strategy is to detect the change. The change that has occurred in the document, in between the times when a resource is visited is the topic of discussion in the subsequent chapters.

4.8 Domain Specific and Incremental Crawling in Migrating Parallel Web Crawler

A migrating parallel crawler consists of several multiple crawling processes which migrate to source of data for example Web server or host before the crawling process is actually started on that Web server or host. Migrating parallel crawlers move to resource and take the advantages of localized access of data. Migrating parallel crawler after accessing a resource migrates to the next host or server or to their home system. Each migrating parallel crawling process performs the tasks of single crawler that it downloads pages from the Web server or host, stores the pages in the local database, extracts URLs from the content of downloaded pages and follows the extracted URLs or links. When Crawling process run on the same local area network and communicate through a high speed interconnection network then it is known as intra-site migrating parallel web crawler. When Crawling process run at geographically distant locations connected by the Internet or a wide area network then it is known as distributed migrating parallel web crawler. The architecture for the domain specific and incremental crawler is proposed in figure 3.1. Here domain specific and incremental crawler is design and implemented, also how the domain specific and incremental crawler operates is explained.

The goals of web crawler are to maintain the local collection fresh and to improve the quality of the local collection. The quality is improved by replacing less important pages with important pages. Thus the web crawler needs to make decision on web page to crawl next. In the algorithm the web crawler makes decisions to crawl a new page, it discards a page from the collection to make space for the new page. This decision of selecting and discarding is known as refinement decision. This refinement decision is based on the importance of pages. To measure importance, the web crawler uses metrics such as PageRank, Hub and Authority. The discarded page should have low importance in the collection in order to maintain the quality. The page of low importance should be replaced by pages of high importance. Along with refinement decision, the web crawler
revisit to update a web page already visited. The web crawler revisits an existing page instead of visiting a new page so that its image can be refreshed. To maintain the collection fresh the crawler has to select the page that will increase the freshness this decision is called as update decision.

The architecture of migrating parallel web crawler consists of modules: Ranking Module, Update Module, Crawl Module and data structures: AllUrls, CollUrls and Collection. AllUrls keeps track of all URLs that the crawler has discovered; CollUrls keep track of the URLs that will be used in the Collection. The URLs in CollUrls are decided by the Ranking Module. The Ranking Module search AllUrls and Collection data structure to make the refinement decision. The web crawler uses Page Rank algorithm for calculation of importance metric. Ranking Module evaluates the Page Ranks of all URLs, based on the link structure.

If a page \( p \) does not exist in the Collection, the Ranking Module estimates the Page Rank of page \( p \), based on how many pages in the Collection have a link to \( p \). The Ranking Module discards the less important page from the Collection to make space for the new page. The Ranking Module replaces less-important page in CollUrls with the more-important page, when a page not in CollUrls turns out to be more important than a page within CollUrls. The URL for this page is placed on the top of CollUrls, so that the Update Module can crawls the page instantly.

The Update Module maintains the Collection fresh based on update decision while the Ranking Module refines the Collection. Update module constantly extracts the top entry from CollUrls and the Crawl Module crawl the page and puts the crawled URL back into CollUrls. The position of the crawled URL within CollUrls is determined by the change frequency of page and its importance. If the URL is closer to the head of the queue it will be visited more frequently.

To estimate a page change, the Update Module records the checksum of the page from the last crawl and compares that checksum with the one from the current crawl. From this comparison, the Update Module decides whether the page has changed or not. The Update Module estimate the change frequency of web page based on change history. For the change frequency of a web page there are two estimators: EB and EP, estimator.
EB is based on a Bayesian inference method while Estimator EP is based on the Poisson process model.

EB categorize pages into different frequency classes, eg: pages that change every week and pages that change every month. While the Update Module records how many times the crawler detects changes to a page for last 6 months. Then EP uses this number to get a confidence interval for the change frequency of that page.

It is possible to keep update statistics on larger units than a web page, such as a web site. If web pages on a web site change at certain frequencies, the crawler may trace down how many times the web pages on that web site changed for last 6 months and get a confidence interval based on the web site-level statistics. Here the web crawler may get confidence interval which is tightly bounded, because the frequency is estimated on larger sample or larger number of web pages. The average change frequency may not be sufficient to determine how often to revisit pages in web site if web pages on a web site change at highly different frequencies.

The Update Module consults the importance of a web page in deciding on frequency of revisit. If a certain web page is most important and the page needs to be updated frequently, the Update Module then revisits the page much more often than other pages with similar change frequency. To implement this case the Update Module needs to store the importance of each web page.

Based on the request from the Update Module, the Crawl Module crawl a web page and saves the web page in the Collection. The Crawl Module extracts all links in the crawled web page and forwards the URLs to AllUrls. If the URLs are new then they are included in AllUrls. Multiple Crawler Modules may run in parallel depending on how fast web pages need to be crawled. The update decision is separate from the refinement decision for high performance. For example, to visit 200 million pages every month, the crawler has to visit pages at 80 pages/second. It is expensive to compute the importance of the web page. To select and deselect web pages for collection is the time taking process. By separating the refinement decision from the update decision, the Update Module focuses on updating pages at high frequency, while the Ranking Module refines
the Collection carefully. The architecture consists of central coordinator system and crawling process.

4.8.1 Central Coordinator System

Central coordinator system consists of Central Crawl Manager, Crawling Application, Web Cache, Central Database or Repository, URL Dispatcher, URL Distributor and Domain Selector. Central crawl manager is the central part of the crawler all other components are started and register with the central crawl manager to offer or request services. It acts as the core of the system. Central Crawl Manager’s objective is to download pages in the order specified by the crawling application. The central crawler has a list of URLs to crawl. After getting the URLs of files, central crawl manager queries the DNS resolvers for the IP addresses of the host or servers.

The central manager then takes into consideration robots.txt files in the root directory of the web server. The central crawler is a set of available crawling process which have registered themselves with the central crawl manager which logically migrated to different domain specific locations. The central crawl manager assigns different URLs to all the crawl process in domain specific manner and in turn the crawling processes begin to crawl the received URL in breadth first fashion and pages are downloaded by multi threaded downloader. It is the responsibility of the crawling application to check for duplicate pages. The crawling application considered in this chapter is a breadth-first crawl strategy. Breadth first search gives high quality pages thus improving the quality of downloaded pages. The DNS resolver uses the GNU adns asynchronous DNS client library to access a DNS server usually collocated on the same machine. A web cache stores web resources in anticipation of future requests. Web cache works on the principle that the most popular resource is likely to be requested in the future. The advantages of Web Cache are reduced network bandwidth, less user perceived delay, less load on the origin servers and quick response time.

When the information resides in web cache, the request is satisfied by a cache, the content no longer has to travel across the Internet. Central database or Repository stores the list of URLs received and downloaded web pages which are collection of the documents downloaded by crawling process. Web pages can be stored in compressed
form. Each document is provided with a unique number called document identifier denoted by doc_id. The documents are stored along with doc_id, length of document and URL. This helps with data consistency and makes data storage much easy and efficient. Data is stored in the central database in hierarchical tree manner.

Each node or the parent node of the tree is the domain and the child node of the parent node is the sub domain. Each node consists of domain name and its corresponding URLs. The database is continuously updated for future use by adding new URL. Central Database communicates with the database of the search engine. URL dispatcher reads the URL database and retrieves the URL from the index table maintained in the central repository and forwards it for crawling. URL distributor classifies the URLs on the basis of domains. It then distributes the URL to the concerned domain specific crawling process for crawling. The URL distributor balances the load on the crawling process. Domain selector identifies the domains of the links which are extracted by the link extractor. The links belonging to specific domain are stored in the corresponding domain table. The domain selector also forwards the domain specific URL to the URL dispatcher for further crawling.

4.8.2 Crawling Process

Crawling process consists of Scheduler, New ordered Queues, Site ordering module, URL Queues/KnownURLs, Multi threaded Downloader, URL Collector, Link Extractor, Link analyzer. Scheduler provides the set of URLs to be downloaded based on the modified page rank. The URLs are stored in new ordered queues. New ordered Queues stores the set of URLs based on modified page rank. Site ordering module provides the modified page rank of the page. KnownURLs are the set of already known URLs also treated as seed URLs. Multi threaded downloader takes a URL from URL collector and downloads the corresponding webpage to store it in the local repository or database. The downloader component fetches files from the web by opening up connections to different servers. URL collector keeps the URL from the downloaded pages. Link Extractor extracts the link from the pages that are downloaded. Link analyzer checks the extracted URLs by the link extractor. If there is similarity in the URL then such URLs are discarded and not further forwarded for processing. When the crawling process runs it requires some memory space to save the downloaded web
pages. Each crawling process has its own local storage. The crawling process saves the downloaded web pages in this database. It is the storage area of the machine on which the crawling process is running. The Ranking Module constantly scans the knownURLs and the local database to make the refinement decision. The Ranking Module refines the local database. The Ranking Module discards the less important page from the local database to make space for the new page. LocallycollectedURLs are the set of URLs in the local collection. The Update Module maintains the local database fresh. In order to maintain the local database fresh, the crawler has to select the page that will increase the freshness most significantly.

4.9 ALGORITHM

Step 1: Start crawling process

Step 2: Register the crawling process with central coordinator system.

Step 3: Wait for domain specific set of URL for crawling.

Step 4: Receive domain specific set of URL from central coordinator system.

Step 5: Add domain specific set of URL to K_URLs (Known URL).

Step 6: Check URL list is empty or not.

Step 7: Read URL from K_URLs.

Step 8: Determine protocol of underlying host.

Step 9: Downloading the document.

Step 10: Call multithreaded download process.

Step 11: Extraction of pages to remove stop word and suffix.

Step 12: Check whether document is downloaded to parse the page and find hyperlink.

Step 13: Check the link present in the known URL and call link analyzer

Step 14: Check the freshness of document and store the document in the local database.

Step 15: Otherwise abandon the document.
Step 16: Calculation of Page Rank

Calculate term weight using TF-IDF;

\[
CS = \text{Content\_Similarity}(); \quad // \text{Calculate content similarity}
\]

\[
PP = \text{Public\_Popularity}(); \quad // \text{Calculate public popularity using server logs}
\]

\[
UF = \text{Update\_Frequency}(); \quad // \text{Calculate site updating frequency}
\]

Final Page Rank = A* CS + B* PP + C* UF;  // where A, B and C are constants

//Compress and filter the downloaded document;

//Send the compressed and filtered content to central database of central coordinator system;

4.10 Crawler Implementation

A client server system based on JAVA as a programming language, Apache TOMCAT as server and MySQL as database has been designed. The implemented model supports all features of the real time III tier architecture. The Net Beans 7.4 is used as Development Platform.

4.10.1 java.lang Package

The basic language functions of java are stored in package called java.lang which is fundamental package. It is used by every java program file. The java.lang is automatically imported into all the programs. The java.lang contains classes and interfaces that are basic to all java programming. The java.lang includes classes such as: Boolean, double, string, Byte, float, system, Character, integer and thread. The interfaces defined by java.lang are Runnable and Cloneable.
4.10.1.1 Runnable Interface

Thread in java is created by instantiating an object of type Thread. The Runnable interface is implemented in order to instantiate an object of Thread type. The Runnable interface abstracts the unit of executable code. The objects that implements Runnable interface, their threads can be constructed. A class need only to implement a single method call run () in order to implement Runnable interface (eg. public void run()).

4.10.1.1.1 Timer Class

Timer class schedule a task for execution at some time in near future. Timer implements runnable interface. A thread is created that waits for a specific time and at scheduled time runs in background. The task linked to the thread is executed when the times arrive. The run () defined by runnable will be executed.

4.10.1.1.1 Chronicle class

Class Chronicle extends Timer class and implements Runnable interface. Chronicle class runs a web crawler in the periodical manner. It makes a Chronicle with two parameter crawler and interval. The parameter crawler is used to run the web crawler in the periodic manner and inter- val parameter is used to invoke web crawler to run after
fixed intervals of time say ‘i’ seconds and is called invocation interval. The web crawler is invoked and it runs every invocation interval. The web crawler is allowed to run for fixed duration and if it is still running it is aborted. There is a process called start chronicling that starts the web crawler instantly using background thread. The process runs the web crawler after fixed intervals of time. The syntax of start chronicling process is public void start (). There is another process called Stop chronicling process, it stops the web crawler if the web crawler is running and the syntax of stop chronicling process is void stop ().

4.10.1.1.2 Crawler Class

The class Crawler implements Runnable interface. Before the web crawler is made to run and start downloading web pages it is initialized. The web crawler is initialized by calling setRoot() function and initializing other crawl parameters.

For every web page and its link a tag or informative label is attached giving information about the page whether the page is home page or a root page. This functionality is carried out by Classifiers. They are registered with addClassifier() function. To monitor the crawl Event listeners are connected. The web crawler is started by calling the function run (). A web crawler in the running state contains two queues. Firstly a priority queue which consist of links that need to be visited, it is a waiting queue and secondly a set of threads that helps in retrieving pages in parallel. The web page downloaded is processed as follows: The web page is passed to classify() method of classifier according to increasing value of priority that is assigned to each web page. For domain specific and incremental crawling the web page is transferred to visit () function of web crawler. Next step is to pass the web page to expand () function of web crawler. When the web page is expanded the web crawler considers only hyperlinks that link to other pages.

The web crawler tests every hyperlink on the web page that is unvisited. The web page is passed through shouldvisit() method. This method approves each web page. The approved web page is then transferred to the crawling queue.
To write a web crawler to an output stream void writeObject (ObjectOutputStream out) method is used. This method is private. Another private method that is used to read a web crawler from an input stream is void readObject (ObjectInputStream in). Method public void run () is used to start the web crawling. The method public void run () implement Runnable interface of the package java.lang. The web crawler runs as the background process. The method that is used to initialize the web crawler for a new crawl is public void clear (). Web crawler is initialized by emptying the crawling queue and setting all the statistics of crawling to zero.

The function that pause the web crawl is public void pause () which is currently in progress and is running. The pause () function first process the current web page which is under processing and then it stops. Web crawler can be restarted by calling run () function again and the content of the crawling queue will remain unchanged. Any thread can call the pause () function. The function that stops the web crawl currently in progress is public void stop (). To stop the web crawler, the current page is first processed then the web crawler is stopped. The crawling queues are initialized to null. The function that is used to timeout the current web crawl in progress is void timedOut (). This function is used internally by the web crawler timer to end the time of web crawl. The web crawler may be any of the different state at any given instant of time. These states are STARTED, PAUSED, STOPPED and CLEARED. The function that returns the state of web crawler is public int getState(), it returns one of the state of the web crawler.

The function that calls for visiting a particular web page is void visit (Page page). Web page is retrieved by the web crawler. The function that decides whether the web page should be traversed or not is a boolean function: public boolean shouldVisit ( ). The function that expand the crawl and explore other links from web page is public void expand ()..

The crawl statistics that the web crawler consider and generates are the number of web pages visited by web crawler, number of links tested by the web crawler, the total number of pages still left to be visited, the number of crawl threads that are currently under execution and number of threads downloading web pages. The number of web pages visited so far in this crawl is obtained by function public int getPagesVisited().
The number of links tested so far in this web crawl is obtained by function public int getLinksTested(). The number of web pages left to be visited is obtained by the function public int getPagesLeft(). The number of threads that are currently under execution and number of threads downloading web pages are obtained by function public int getActiveThreads().

4.10.1.2 Cloneable Interface and clone ()

In Clone () method the duplicate copy is generated of the object on which it is called. Only those classes can be cloned that implements the Cloneable interface. A clone is the exact copy of the original. For a class to be cloned, a bit wise copy of the object is made. However, an exception CloneNotSupportedException is thrown if clone () method is called on class that does not implement interface Cloneable.

4.10.1.2.1 Download Parameter Class

A web crawler has certain number of download parameters: number of background threads used by the web crawler, maximum size of page in Kbytes, timeout time for a single web page measured in seconds, timeout time for entire web crawl measured in seconds and obey crawling rules in robots.txt; a robot exclusion file. The class DownloadParameters declared as public implements Cloneable interface. The download parameters are upper bound on how web crawler can download a Link. The function that make a DownloadParameters object with default settings is public DownloadParameters (). The function that clones a DownloadParameters object is public Object clone ().

4.10.2 HTMLTransformer Class

HTMLTransformer Class makes an HTMLTransformer that writes pages to a stream as well as to a file.
4.10.2.1 LinkTransformer Class

LinkTransformer Class is a class that is a derived from super class HTMLTransformer. It is a Transformer that maps URLs into links. The default function of LinkTransformer is that it transforms all links into absolute URLs. LinkTransformer class makes a LinkTransformer that writes web pages to a file or stream.

4.10.2.1.1 RewritableLinkTransformer Class

RewritableLinkTransformer class is derived from LinkTransformer class. If the mapping of URL changes as HTML has been changed, the RewritableLinkTransformer remaps URLs in such a way that HTML can be corrected. Concatenator and Mirror classes use RewritableLinkTransformer class because the URL mapping function changes in these operations as each web page is written to the concatenation or mirror.
4.10.2.1.1 Concatenator Class

Concatenator class is derived from the RewritableLinkTransformer Class. The transformer that concatenates multiple web pages into a single web page is called as Concatenator. The complete sets of concatenated web pages are wrapped between prolog and epilog. Prolog and epilog are constant strings of HTML. Also each web page is wrapped between header and footer. Adjacent web pages are delimited by a divider. The transformations that Concatenator performs on web pages before appending them together are: Concatenator deletes the conflicting elements including HTML tags such as: HEAD, TITLE, BODY, HTML, STYLE and FRAMES. The Concatenator changes links that are present in the written web pages into in-page references, denoted by "#concatenator_N". The Concatenator changes links to other web pages into absolute references.

4.10.2.1.2 RecordTransformer Class

RecordTransformer class is derived from RewritableLinkTransformer class. It consist of following functions void setProlog (String prolog), String getProlog (), setEpilog (String epilog), String getEpilog (), void setRecordStart (String recordStart), String getRecordStart (), void setRecordEnd (String recordEnd), String getRecordEnd ().

4.10.2.1.2 Mirror Class

Mirror class is derived from LinkTransformer class. When the web pages are mirrored, the files are stored on local disk mirroring their URLs in a proper directory structure. Mirror is website stored offline. Mirror class gets the filename used for directory URLs.

4.10.3 CrawlListener Interface

There are several events that need to be notified such as web crawler has started, exhausted its links, cleared its state, timed out and paused. To notify each of these
events there are several abstract functions. The function public abstract void started (CrawlEvent event) is used to notify that web crawler has started. The function public abstract void stopped (CrawlEvent event) notifies that web crawler has exhausted its links. The function public abstract void cleared (CrawlEvent event) is used to notify that web crawler has cleared its state. The function public abstract void timedOut (CrawlEvent event) notifies that web crawler has timed out. The function public abstract void paused (CrawlEvent event) notifies that web crawler is in paused state.

4.10.3.1 CrawlAdapter

CrawlAdapter class implements CrawlListener interface. CrawlAdapter is an Adapter for interface CrawlListener. The function public void started (CrawlEvent event) is used to notify that web crawler has started. The function public void stopped (CrawlEvent event) notifies that web crawler has exhausted its links. The function public void cleared (CrawlEvent event) is used to notify that web crawler has cleared its state. The function public void timedOut (CrawlEvent event) notifies that web crawler has timed out. The function public void paused (CrawlEvent event) notifies that web crawler is in paused state.

4.10.4 LinkListener Interface

Link event listener is an interface LinkListener. The function that notifies about occurrence of event on a link is public abstract void crawled (LinkEvent event).

4.10.4.1 EventLog Class

EventLog implements two interfaces namely CrawlListener and LinkListener interface. Test is performed to know whether network related Link Events are printed by the logger. If the value is true then LinkEvents are printed by the logger. If the value is false then all link events are not printed by the logger. True is the default value.
EventLog notifies the state of the web crawler such as web crawler has started, web crawler has stopped, the state of web crawler is cleared, web crawler has timed out, web crawler has paused and that a link event had occurred.

4.10.5 java.io.Serializable

The process when the state of the object is written to a byte stream is called serialization. The process is beneficial to save the state of program to permanent persistent storage example a file. When need arise these objects may be restored by using the reverse process of serialization i.e., deserialization. No members are defined by serialization interface. If a class is serializable then all of its subclasses are serializable. An object implementing the serialization interface can be saved and stored by the serialization facilities.

Figure 4.3 Class Diagram (java.io.Serializable)
4.10.5.1 Action Interface

Interface Action extends java.io.serializable. It consists of following functions namely:
void connected (Crawler crawler), void disconnected (Crawler crawler) and void visit
(Page page).

4.10.5.2 PagePredicate Interface

Interface PagePredicate extends java.io.serializable. It consists of following functions
namely: void connected (Crawler crawler), void disconnected (Crawler crawler) and
Boolean shouldActOn (Page page).

4.10.5.3 Classifier Interface

Public Classifier interface extends java.io.serializable. A classifier is an object that
annotates links and web pages with labels. It uses two functions namely Link.setLabel()
and Page.setLabel(). When a web page is retrieved by the web crawler, first and the
foremost it is transferred to classify () method of every classifier that is registered with
web crawler. Classifiers classify links into categories like parent or child. Classifiers
classify pages into different categories like biotechnology or medical or computer
science and parsing formatted pages in style of Yahoo, AltaVista or latex2html.

In order to classify a web page the classifier calls two functions namely page.setField()
and page.setLabel() to mark up the web page. For the execution of priority, lower
priorities are executed first. Classifier function used to get the priority is public float
getPriority (). A classifier defines priority which is constant and is declared public so
that classifiers can compute their priorities that depend on it. The classifier consider the
link of the page and call the function link.setLabel() to mark up links.
4.10.5.4 LinkPredicate Class

Interface LinkPredicate declared as public is derived from java.io.Serializable. It consists of several functions namely: void connected (Crawler crawler), void disconnected (Crawler crawler), boolean shouldVisit (Link link).

4.10.6 Databases

Web Crawler has been implemented by using MySQL databases. For the connection to establish firstly there must be a MySQL JDBC driver in the ./lib directory. MYSQL JDBC driver consists of .jar files that are needed to establish the connection. The class name of the MySQL JDBC driver must be specified in file crawler.properties. A script file called as databasescript.txt has been created that can be loaded by MySQL to create the database and tables. All kinds of SQL queries are made to run on the database using MySQL client.

4.10.6.1 Database for URL Visited

MySQL Server contains a database with name 'webcrawler' and a table with name 'urlvisited'. The table urlvisited have following attributes:

1. crawlid int(10) unsigned Primary Key
2. url varchar(255) Primary Key
3. crawler int(10) unsigned
4. urlnr int(11)

4.10.6.2 Database for Crawled Documents

MySQL server contains a database with name ‘codecrawler' and a table with name 'documents'. The table documents have following attributes:

1. url varchar(255) Primary Key
2. crawlid int(10) Primary Key
3. context varchar(10)
4. type varchar(10)
5. crawldate date
6. page mediumtext

Attribute 'page' contains all crawled and stored documents and can be used for analyzing content offline with another application.

### 4.10.6.3 Database Schema

The snapshot shown below is the database schema.

![Database Schema](image)

#### Figure 4.4 Database Schema

### 4.10.7 Connection between MySQL Database and Java

The MySQL database is connected with the Java with the help of MySQL driver. For connection to be made with database an account is created in MySQL. The data is
accessed or retrieved from MySQL database once a connection is established. The following code establishes the connection between java files and MySQL database. Once the connection with the database is established the message shown is "Connected to the database", if there is no connection with the database then the message displayed is "Disconnected from database". The interface in java.sql package that specifies connection with specific database technologies such as Ms-Access, Oracle, MySQL and java files is called Connection. The method that attempts to load the class and returns class instance and after that matches class with given string is Class.forName (String driver). The method is static in nature. It takes string type value (driver). DriverManager is a class of java.sql package. Class DriverManager controls a set of JDBC drivers which are used for establishing the connection. It is mandatory for every driver to get register with DriverManager class. The method that establishes a connection to specified database url is getConnection(String url, String username, String psswd). This method uses three string types of arguments: url: -is the url of database stored, username: - name of user of MySQL, psswd: -Password of user of MySQL.

The method that is used for disconnecting the database connection is con.close(). The method frees all the resources occupied by the database. The method that shows error messages is printStackTrace().

```java
import java.sql.*;
public class MysqlConnect{
    public static void main(String[] args) {
        System.out.println("MySQL Connect.");
        Connection conn = null;
        String url = "jdbc:mysql://localhost:3306/";
        String dbName = "jdbc";
        String driver = "com.mysql.jdbc.Driver"; //jdbc driver
        String userName = "root"; //username
        String password = "root"; //password
    }
}
```
try { //exception Class.forName(driver).newInstance();
    conn = DriverManager.getConnection(url+dbName,userName,password);
    System.out.println("Connected to the database"); conn.close();
    System.out.println("Disconnected from database"); } catch (Exception e) {
    e.printStackTrace(); } }

The figure 4.5 is the Multithreaded Server; the figure 4.6 is the Client Crawler process. The crawling process migrates to different machines to download the content. The figure 4.7 is the Migrating Parallel Web Crawler crawling Integral University Pages. The figure 4.8 is the Migrating Parallel Web Crawler crawling Banaras Hindu University Pages. The figure 4.9 is the Migrating Parallel Web Crawler crawling Aligarh Muslim University Pages. The figure 4.10 is the Migrating Parallel Web Crawler crawling KNIT, Sultanpur Pages.
Figure 4.5: Multithreaded Server

Figure 4.6: Client Crawler
Figure 4.7: Migrating Parallel Web Crawler crawling Integral University Pages
Figure 4.8: Migrating Parallel Web Crawler crawling Banaras Hindu University Pages
Figure 4.9 Migrating Parallel Web Crawler crawling Aligarh Muslim University Pages
Figure 4.10: Migrating Parallel Web Crawler crawling KNIT, Sultanpur Pages
4.11 Result

The table 4.3 shows basic crawl statistics, table 4.4 shows HTTP errors that were generated during the crawling process. The figure 4.11 is the graphical representation of the HTTP errors.

<table>
<thead>
<tr>
<th>HTTP Requests</th>
<th>51,743,856</th>
</tr>
</thead>
<tbody>
<tr>
<td>network errors</td>
<td>1,885,652</td>
</tr>
<tr>
<td>read_timeout_exceeded</td>
<td>732,892</td>
</tr>
<tr>
<td>robot.txt requests</td>
<td>5,424,086</td>
</tr>
<tr>
<td>successful non robots</td>
<td>42,785,652</td>
</tr>
<tr>
<td>average page size</td>
<td>10548 bytes</td>
</tr>
</tbody>
</table>

Table 4.3: Basic Crawl Statistics

<table>
<thead>
<tr>
<th>Total successful non robots</th>
<th>42,785,652</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 (OK)</td>
<td>36,795,661</td>
<td>86%</td>
</tr>
<tr>
<td>404 (not found)</td>
<td>2,236,965</td>
<td>5.22%</td>
</tr>
<tr>
<td>302(moved temporarily)</td>
<td>1,786,452</td>
<td>4.17%</td>
</tr>
<tr>
<td>301 (moved permanently)</td>
<td>985,985</td>
<td>2.30%</td>
</tr>
<tr>
<td>403(forbidden)</td>
<td>136,987</td>
<td>0.32%</td>
</tr>
<tr>
<td>401(unauthorized)</td>
<td>136,547</td>
<td>0.31%</td>
</tr>
<tr>
<td>500(internal server error)</td>
<td>29,856</td>
<td>0.06%</td>
</tr>
<tr>
<td>others</td>
<td>35,254</td>
<td>0.08%</td>
</tr>
</tbody>
</table>

Table 4.4: HTTP errors
The implementation was made to run on .org, .com, .info, .edu, .jobs, .biz sites along with conventional crawler, single threaded, agent based crawler. The results obtained are given below in table 4.5.

<table>
<thead>
<tr>
<th>Crawler Type</th>
<th>.org</th>
<th>.com</th>
<th>.info</th>
<th>.edu</th>
<th>.jobs</th>
<th>.biz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional Crawler</td>
<td>12</td>
<td>14</td>
<td>21</td>
<td>30</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>Single threaded Crawler</td>
<td>10</td>
<td>22</td>
<td>22</td>
<td>21</td>
<td>21</td>
<td>25</td>
</tr>
<tr>
<td>Agent Based Crawler</td>
<td>14</td>
<td>18</td>
<td>22</td>
<td>21</td>
<td>21</td>
<td>32</td>
</tr>
<tr>
<td>Migrating Parallel Crawler</td>
<td>24</td>
<td>26</td>
<td>35</td>
<td>38</td>
<td>25</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 4.5: Domain specific and Incremental crawling
The basic crawling statistics that are generated when the migrating parallel crawler was executed, total HTTP requests made are 51,743,856 out of which network errors were 1,885,652, time out errors are 732,892, robot.txt request are 5,424,086, successful non robot request are 42,785,652, the average page size is 10458 bytes. About 86% of the links were explored while 5.22% of the links were not found. The others HTTP errors and their percentage are moved temporarily 4.17%, moved permanently 2.30%, forbidden 0.32%, unauthorized access 0.31% and internal server error 0.06%. Table 4.5 shows the crawl data of the conventional crawler, single threaded crawler, agent based crawler and proposed migrating parallel web crawler. Each of the crawlers was executed to collect domain specific and fresh pages. The above figure 4.12, indicate that the implementation improves domain specific crawling and incremental crawling of web pages. The implementation collected more domain specific pages of .org, .com, .info, .edu, .jobs, .biz sites when compared with conventional crawler, single threaded and agent based crawler.
4.12 Conclusion

In this chapter an Extended Model for Effective Migrating Parallel Web Crawling with Domain Specific and Incremental Crawling is proposed and implemented. The crawled pages are domain specific. The domain specific and incremental crawling is performed in migrating parallel web crawlers compared with the other crawling techniques. As change frequency is considered fresh pages are downloaded. Web crawling in breadth first manner ensures that high quality of pages is downloaded. Breadth first crawling approach tends to improve the quality of downloaded pages. Domain specific crawling will yield high quality pages. The crawling process is migrating to host or server with specific domain and downloading pages within specific domain. Incremental crawling keeps the pages in local database fresh thus increasing the quality of downloaded pages.