CHAPTER 4
A NOVEL PARALLEL DOMAIN FOCUSED CRAWLER

4.1 Introduction

Crawler mobility allows more sophisticated crawling algorithms and avoids some of the inefficiencies associated with the brute force strategy used by the current crawlers. The mobile crawling is an efficient, scalable solution to establish a specialized SE in the highly distributed, decentralized and dynamic environment of the Web. The mobility is defined in the context of Web crawling as the ability of a crawler to transfer itself to a Web server of interest before collecting a webpage on that server. After completing the collection process on a particular Remote Server (RS), the crawler together with the collected data moves to the SE. Different approaches (Fiedler and Hammer 1999; Hammer and Fiedler 2000; Papapetrou and Samaras 2004) were proposed and many systems were suggested to build the Web indices based on Mobile Crawlers (MC) powered by mobile agents (Nwana 1996). The MC can be transported to the site of the source where the data resides in order to filter out any unwanted data locally before transmitting it back to the SE. These MC can reduce the network load caused by the crawler by reducing the amount of data transferred over the network. The Internet traffic and load on the resources (such as Bandwidth, CPU cycle and Memory) can be reduced only if the crawler accesses only those webpages that are actually modified.

This chapter proposes a new MC architecture and an alternative approach of URL allocation and filtration of the webpages. This approach assists the SE to perform efficient URL allocation to the MCs and filters out all those webpages at the SE, which have low probability of change. These crawlers then move to the Remote Server (RS), filter those webpages, which have not been modified
since the last crawl and send the modified webpages to the SE, after compressing them. Here the emphasis is on parallelizing the crawling process so that it can maximize the download rate while minimizing the load on the network.

4.2 Problems with the Existing Traditional and Mobile Crawling Techniques

To build a comprehensive full text index of the Web and keep it up-to-date, the crawlers consume a significant number of resources of the underlying network. According to the study of (Cho et al. 1998), it is estimated that Web crawlers of commercial SE crawl up more than 10 million webpages per day.

The major problem with the traditional crawling technique is that there is no way to determine the webpage that has actually been modified since the last crawl at the RS. Whatever can be done is at the SE. This only results in wastage of resources in indexing the webpages that have already been indexed by the SE.

Many studies (Fiedler and Hammer 1999; Hammer and Fiedler 2000; Papapetrou and Samaras 2004; Abkenari and Selamat 2010; Nath and Bal 2011) suggested the mobile crawling technique to overcome this problem. In these techniques, the MC crawls the webpages and stays in the RS to monitor any change that can take place in the webpages allotted to it, but these techniques have their own problems such as:

(i) The MCs that always stay in the memory of the RS occupy a considerable portion of its memory. This problem is further aggravated, when more MCs from different SEs stay there. This can produce scarcity of memory on RS because RS have to perform their own task also. All these MCs stay in the memory of the RS and consume a lot of memory that could have otherwise been used for some other useful purposes.

(ii) It can also happen that the RS may not allow the MCs to reside permanently in its memory due to security reasons.
(iii) If a webpage changes rapidly and every time a MC catches these changes and send these changed webpages to SE, then it again consumes more resources of the network including the processing of the system.

(iv) The MC proposed by (Nath and Bal 2011) was the best crawler reported in literature. It considered following three parameters i.e. (i) Change due to change in URLs (ii) Change due to change in number of keywords (iii) Webpage retrieved directly. The limitation of PMCS is that it does not consider any changes in textual contents of webpage. So, if even any webpage is being changed everyday then PMCS won’t be able to detect these changes.

To address these problems, an architecture for mobile crawling has been proposed that makes the use of frequency change estimator at the SE for efficient URL allocation and filtering out those webpages at the RS that have not actually been modified since the last crawl.

4.3 The Architecture of the Proposed Parallel Domain Focused Crawler

The MCs are constructed as mobile agents and are dispatched to RS. They download the webpages, process them and compare the old and new ASCII character count of the extracted webpage. If the webpage is found modified then it is compressed and transmitted back to the SE. The proposed architecture is able to filters the pages at two places – first at the SE during URL allocation and secondly at the RS during crawling. The architecture of the proposed PDFC is shown in Figure 4.1. The major components of proposed PDFC are: Analyzer Module (AM), Old Database File (ODF), Link Extractor (LE), Domain Name Director (DND), Crawler Hand (CH), Crawler Manager (CM), Statistics DataBase (SD), Frequency Change Estimator (FCE), Remote Server (RS) and the Comparator (COM). Each of these components is discussed in detail below.
**Crawler Manager (CM):** The main tasks of CM are generation of MC, allocation of URLs to the MC for crawling the webpages based on the information taken from the Frequency Change Estimator, construction and updation of the data in ODF, updation of Statistics Data Base, receiving the crawled webpages from the RS and decompressing and indexing them. After generation and URL allocation to the MCs, these MCs are sent to the designated RS for crawling.

**URL Dispatcher:** URL dispatcher receives the extracted URL from AM and forwards them to URL Queue in RS.

**Statistical Database (SD):** This module is maintained in the SE and contains information about all the webpages crawled by the MC. The database maintains the following six fields:

a) **NAME_OF_URL:** It stores the name of the webpage that is indexed by the SE in its database after downloading.

b) **PARENT_URL:** It depicts the original URL entered by the user corresponding to which the URLs in the NAME_OF_URL are fetched. For example, if two URLs i.e. (i) http://xkcd.com/ and (ii) http://dynamic.xkcd.com/random/comic are available then URL entered by the user is http://xkcd.com/ serves as the PARENT_URL and http://dynamic.xkcd.com/random/comic serves as NAME_OF_URL.

c) **ASCII_CHARACTER_COUNT:** It is a numeric value which is calculated by summing the ASCII value of each character available in the webpage. It is computed whenever the webpage is downloaded.
AM: Analyzer Module  
CM: Crawler Manager  
SD: Statistics Data Base  
LE: Link Extractor  
ODF: Old Database File  
COM: Comparator  
URL Disp.: URL Dispatcher  
WWW: World Wide Web  
DND: Domain Name Director

Figure 4.1: Proposed Parallel Domain Focused Crawler
d) LAST_MODIFIED_DATE: It refers to the date when the last time a particular URL had been crawled and found changed at the RS. This field may be empty for many webpages since many RS do not maintain LAST_MODIFIED_DATE.

e) FREQUENCY_OF_CHANGE: This field stores the frequency of webpage change for each webpage and is updated by the FCE module after every crawl. The value of this field is used by the CM during URL allocation to the MC to filter the webpages whose probability of change is small. The information stored is updated by the CM after each crawl by analyzing the webpages received from the MC.

f) FILE_PATH: The location of the HTML file stored in the repository of SE is reflected in this field.

**Old Database File (ODF):** ODF contains the statistics about each webpage to be crawled. The fields of this module are NAME_OF_URL and ASCII_CHARACTER_COUNT. The CM constructs one ODF for each MC. The values of these fields are taken from SD. All the values of this module is sent with the corresponding MC to the RS. This is a temporary module and is destroyed after the MC completes the task of crawling. Every time during the URL allocation this module is created afresh by the CM, one for each MC.

**Comparator Module (COM):** The COM checks whether the downloading period of a webpage has reached or not? If yes, then send the URL of this webpage to AM otherwise rejecting the URL. This is achieved by comparing the statistics of each webpage taken from ODF with the statistics collected by the AM from the corresponding webpage at the RS to filter out the webpages that are not modified. The COM first compares LAST_MODIFIED_DATE and FREQUENCY_OF_CHANGE, taken from the ODF that were computed when the URL was previously crawled. During this comparison if the difference between the current date and the last-modified date is less than or equal to the
value of FREQUENCY_OF_CHANGE, then the new URL is not processed rather, it is ignored. On the other hand, in case the difference between these two is greater than the value of FREQUENCY_OF_CHANGE, then webpage is processed. This module works on both sides i.e. SE and RS. At the RS it takes current value of ASCII_CHARACTER_COUNT and compares it with the old value of the same webpage. If both are different then it means the webpage is updated and sends it to SE, otherwise the webpage is rejected. Thus, it helps in deciding whether the retrieved webpage should be sent to SE or not.

**Analyzer Module (AM):** It works on SE as well as RS. At the SE, it constructs SD. Initially, it creates this data by scanning and analyzing each webpage crawled by the MC and then updates this data after every crawl. Then it is sent to the RS with the MC and a copy of it is stored in the secondary memory of the RS for future use by the MC.

**Link Extractor (LE):** LE receives and decompresses the downloaded Webpage coming from the RS to SE. It extracts all the URLs available in the downloaded webpage. Then these URLs are given to the COM module to check whether they are previously visited or not.

**URL Queue:** The URLs coming from the SE are received by the URL Queue at the RS. DND retrieves URL from this queue and place them in their corresponding Domain Queue.

**Domain Name Director (DND):** The job of DND is to select the specific domain for the URLs that are coming from the URL queue and forward it to the particular DNS Queue (such as .org, .com, .edu etc.). Given a URL to the DNS Queue, the number of webpages to be downloaded and the required time depend upon the URL itself. The distribution of the load on the CH is likely to be different depending on the frequency of demand of the domain.
**Frequency Change Estimator (FCE):** This module identifies whether two versions of a webpage are same or not and thus helps to decide whether the existing webpage in the repository should be replaced with the changed one or not. This module is the part of the SE and remains there. The FCE is used to calculate the probability of the webpage change. It means in how many days a webpage gets changed. This module maintains the webpage change frequency of every webpage at the SE and updates this information in SD every time a webpage is crawled. This information about the webpages helps the SE in deciding that which webpage is to be re-crawled and when. The CM uses this information for URL allocation to the MCs. This information helps the CM in deciding which URLs are delegated and when to the MC. Thus, the FCE filters out all those webpages at the SE that have low probability of change and reduces the work load of the crawler on the network. To judge these changes two assumptions made here are: (i) webpage change between accesses is known; (ii) Webpage changes are accessed at regular intervals.

To track the probability of webpage change a variable called Frequency Change Estimation (F) is used in the proposed PDFC. The values of F are defined as follows:

F = -1 (for those URLs, which are traversed for first time)
F = 0 (for those webpages, which changes at rapid rate, so frequency parameter cannot be applied for such webpages)
F = N (N>0; N= number of days since last change of webpage occurred)

The above notations are explained as below:

(i) If the value of F = -1, shows that the URL is not previously visited. Hence the URL should be crawled in order to get updated information on the SE end. Once the webpage arrives at the SE then the AM scans and analyzes the webpage in order to get its statistics.
(ii) If the value of F = 0, then it assumes that the information about the webpage change is not available because the webpage is frequently changed and it is required to be processed the URL again in order to refresh the information available to SE. Furthermore, there is no need to send these webpages to AM for scanning and analysis to collect the statistics about them in SD.

(iii) If the value of F=N, shows the number of days in which the webpage is to be re-crawled. This is analyzed based on the statistics data available in SD. The difference between current date and last modification date is calculated. If the value of the difference is less than or equal to the value of F then no processing is done on this URL. If the difference is greater than the F then the URL is forwarded to the ODF for downloading the webpage. Whenever the exact time of frequency of change occurs then the corresponding URL is assigned to the MC for downloading.

**Crawler Hand (CH):** CH takes the URL from DNS Queue and receives the corresponding webpage at the RS. It then decides whether the webpage is to be send to the SE or not. The expansion of this module is shown in Figure 4.2 where the CHs got URLs as an input from DNS Queue and store them in URL receiver. Then Crawling Process (CP) is generated by the MC which retrieves the webpage from RS. These webpages are further sends to the Manager. The Manager calculates the corresponding ASCII_CHARACTER_COUNT of that webpage with the help of ASCII calculator. Then Manager sends old ASCII_CHARACTER_COUNT (OAC) and newly computed ASCII_CHARACTER_COUNT (NAC) to COM module. COM compares both of these values. If these values are same then it rejects the webpage otherwise it sends this webpage to SE after compression. The CHs are involved in actual downloading of the webpages. They depend on DNS Queue on which they settled for getting the URLs. CP Director takes URL from the URL receiver and receives the corresponding webpage from RS. The MC can run multiple CPs at a time, so they can share the load in between different processes.
Each CH is independent of other; it means they have no communication with each other because the crawling hand gets the seeds URL from the independent and respective DNS Queue only. The number of CP supported by the architecture is not fixed. It may be scaled up or down depending on availability of resources and actual requirement.

![Diagram](image_url)

**Figure 4.2: Expanded Architecture of Crawler Hand**

The CPs are capable enough to handle the webpages, which are not allowed to be crawled. It also automatically discards URLs that are in DNS Queue but exist no more on WWW. Therefore, a CP, before starting the actual downloading of webpages, checks for its actual existence on the Web using the RS response. If there is no response from the RS or the webpage is prohibited for traversing as indicated by the corresponding ROBOT.TXT file, it immediately discards the
URL without waiting for further responses in order to save the network bandwidth.

**ASCII Character Comparator:** The ASCII Character Comparator catches the situation where the webpage’s contents are changed from reader’s point of view. So, it is desirable to calculate one or more character changes. As in any educational institute the students are changed year by year. But sometimes the change in name of student is very minor may be limited to very few characters.

Figure 4.3(a) shows initial HTML webpage source code having a name Mahesh doing Ph. D. under the guidance of Dr. XYZ. Suppose this information was obtained from the ABCD University Website (say in 2009).

```html
<html>
<title>ABCD University</title>
<head><b>Department of Computer Science and Application 2009</b></head>
<body>
<center>
Mahesh doing Ph. D. under the guidance of Dr. XYZ
</center>
</body></html>
```

Figure 4.3(a): Initial Version of Webpage

Figure 4.3(b) shows the changed version of webpage where after two years (say in 2011) this webpage was got changed and all the contents of the webpage were found similar except three characters of name and last two digits of the year.

```html
<html>
<title>ABCD University</title>
<head><b>Department of Computer Science and Application 2011</b></head>
<body>
<center>Saresh doing Ph. D. under the guidance of Dr. XYZ</center>
</body></html>
```

Figure 4.3(b): Changed Version of Webpage
All this happens because Mahesh has passed his Ph. D. program. So, the PDFC is able to catch such a minor structural changes also.

4.3.1 Working of PDFC

The proposed architecture of PDFC is explained in this section. The working of PDFC is divided into two cases.

Case1: When PDFC runs for the first time.
Case2: When PDFC starts re-crawling the Web.

Each of these cases is explained with sufficient detail given below:

**Case 1: When PDFC runs for the first time:** To initiate the crawling process, a seed URL is provided to the CM module in the PDFC. The CM considers it as a new URL and generates an MC for download the webpage related by the URL, URL Dispatcher then sends it to the RS where it is shown in URL Queue. The DND takes the URL from the URL Queue and determines the domain of the URL by using Domain Name Server and stores it in the Domain Name Server Queue (DNS Queue) corresponding to the domain. Numbers of DNS Queue are equal to the number of domains. The CH then takes the URL from DNS Queue and determines IP address of the RS that contains the webpage. Then CP requests the RS to provide the ROBOT.TXT file to it and checks the file for the download permissions. If it has no permission to access the specified URL then the CP rejects the URL otherwise the protocol for communication between the CP and RS is decided. The CP retrieves the webpage corresponding to the URL by using the selected protocol and sends it to the Manager. The Manager forwards this webpage to the ASCII Character Comparator for further processing.

When there is no ODF associated with the MC indicates the URL is new and simply the webpage is compressed by ASCII Character Comparator and sends it to the LE of the SE. The LE decompresses the webpage, extracts the URLs (if any) from the webpage and sends them to the COM module. The decompressed webpage is also forwarded to the SD for storage.
Case 2: When PDFC starts re-crawling the Web: Once the process of crawling is initiated, as discussed in Case 1. The URLs extracted from the first webpage are checked whether they are already traversed or not. When they are not traversed earlier then Case 1 is repeated. When they are traversed then they are forwarded to COM. COM checks for their freshness period is reached or not. If freshness period is reached then CM generates MC and sends the URL with its ODF to the URL Dispatcher. URL Dispatcher dispatches the coming URL to the URL Queue at RS. DND takes this URL and places it into its corresponding DNS Queue. The webpage corresponding to the URL is downloaded and given to Manager. The Manager processes it in the same way as used in Case 1. This webpage is sent to the ASCII Calculator, which calculates the NAC of this webpage and returns it to the Manager. Now Manager sends OAC and NAC to the ASCII Character Comparator to compute their equality. If both are found equal then the webpage is rejected otherwise MC compresses the webpage and sends it to the LE at SE. This process is repeated until DNS Queue becomes empty. The complete working of the CP is represented with the help of an algorithm shown in Figure 4.4.

The complete Indexing process of a webpage is given in algorithm shown in Figure 4.5. Furthermore, the flowchart corresponding to the working of PDFC is shown in Figure 4.6. The MC used in PDFC carries the ODF, the COM and the AM with it and goes to the RS from where the webpages are to be crawled.
**Algorithm:** Crawling Process

**Input:** URLs from DNS Queues

**Begin**

Step 1: Takes the URL from DNS Queue and IP address of RS from Domain Name Server

Step 2: Fetch the ROBOT.TXT file from the RS

Step 3: Decide the protocol (HTTP, FTP, TELNET etc.) to be used for Communication between CP and RS

Step 4: Retrieve the webpage corresponding to the URL by using the Protocol

Step 5: Calculate the NAC for the retrieved webpage

Step 6: **If** (OAC = NAC of the URL) **Then**

Reject the webpage

**Else**

Compress and download the webpage

Step 7: **Repeat** Steps 1 to Step 6 **until** DNS Queue is empty

**End**

**Output:** Downloaded webpages

---

**Figure 4.4: Algorithm for Crawling Process**

The MC accesses those webpage one by one whose URLs are given in the ODF, calculates the NAC of the webpage and compares it with OAC of the same webpage. If both are equal then this webpage is rejected. Otherwise, the webpage corresponding to the URL is sent to the SE for updating the information. The LAST_MODIFIED_DATE may not be available for all the webpages, so all the webpages whose last modification date is not available, are directly downloaded and sent to the SE for updation without any comparison of ASCII_CHARACTER_COUNT at the RS.
Algorithm: Webpage Indexing

Input: Seed URL(s) provided by the user

Begin

Step 1: URL Dispatcher sends the seed URL to URL Queue

Step 2: Distribute URLs based upon their domain name into DNS Queue

Step 3: While DNS Queues are not empty repeat the Step from 5 to 9

Step 4: CP fetches the webpage from the RS by taking URL from DNS Queue

Step 5: ASCII Calculator Calculates the NAC for the webpage

Step 6: ASCII Character Comparator compares OAC and NAC

Step 7: If (OAC = NAC) Then

Reject the webpage and goto Step 4

Else

Compress the webpage and send it to LE at SE

Step 8: LE decompresses the webpage, send it to SD, extract the URLs from the webpage and send them to the COM

Step 9: COM checks the URL in SD for its existence

Step 10: If (URL is present in SD) Then

If (URL present in ODF) Then

Forward this URL to AM

Else

Reject the URL and Exit

Else

Directly send the URL to URL Dispatcher and goto Step 4

Step 11: AM sends the URL with its ODF to URL Dispatcher and goto Step 2.

End

Output: Downloaded webpage

Figure 4.5: Algorithm for Indexing of Webpages
Figure 4.6: Flow Chart for Indexing of Webpages
4.4 Performance Evaluation of PDFC

The PMCS developed by (Nath and Bal, 2011) was the best crawler reported in literatures; therefore the proposed PDFC is compared with the PMCS. The following parameters have been used to compare the proposed PDFC with the existing PMCS mobile crawler.

(i) Webpage Change Behavior
(ii) Load on the Network
(iii) Time Consumption
(iv) Network Utilization

4.4.1 Webpage Change Behavior: The authors have assumed that a webpage is changed if the ASCII_CHARACTER_COUNT of the web is increased or decreased. This webpage change behavior is obtained by identifying the frequently changed webpages and less frequently changed webpages. The webpages that are found changed on every crawl are called frequently changed webpage and the webpages that are found changed after an interval (not at a specific time) are called less frequently changed webpage.

4.4.2 Load on the Network: The load on the network is the amount of data transfer from the RS to the SE through the network and is measured in KBs. When the MC retrieves the webpages from the RS, it causes load on the underlying network. This needs to be reduced because the network is not expanding as compared to the growth of the Web. The average load on the network (l) is calculated as:

\[ l = (p \times s) + \text{Overhead} \quad \text{... (4.1)} \]

Where

‘p’ represents average number of webpages retrieved
‘s’ is average size of each webpage
Overhead is the extra load due to the extra modules.
4.4.3 Time Consumption: The average time consumed (t) for crawling and
downloading of webpages is computed by using equation given below:

\[ t = \frac{l}{r} \]  \quad \text{... (4.2)}

Where ‘l’ is the load on the network and ‘r’ is data transfer rate of the network.

Performance on Packet Preservation: Performance of PDFC and PMCS is
also evaluated in terms of average packet preservation and is computed by the
equation given below:

Packet Preservation = Packets sent using PMCS – Packet sent by using PDFC
\quad \text{... (4.3)}

4.4.4 Network Utilization

Network utilization is the ratio of current network traffic to the maximum traffic
that a port can handle. It indicates the magnitude of the bandwidth being used in
the network. High network utilization indicates that the network is busy, low
network utilization indicates that the network is idle. When network utilization
exceeds the threshold under normal condition, it causes low transmission speed
and delay.

4.4.5 Experimental Setup

PDFC was implemented in Java using JDK 1.6 version and Netbeans IDE 7.0.1.
MS Access was used for storing the data. One hundred webpages (See appendix
A) were selected to perform the experiment. Initially, these webpages were
downloaded. The total size of the webpages was 800 KBs and hence the average
size of each webpage was 8KBs. These webpages were regularly visited twice a
day over a period of 30 days. The following types of data were collected during
the experiment of PDFC.
(i) Number of webpages that were changed frequently.
(ii) The numbers of webpages that were changed less frequently.
(iii) The number of webpages which were not modified.

In order to compare the proposed PDFC with the existing PMCS (Nath and Bal 2011), the PMCS was also implemented using the same technology and same inputs were given to it. The experiment was performed for same period and above mentioned data was collected. Table 4.1 shows data collected for PDFC and PMCS.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>PDFC</th>
<th>PMCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Frequently Changed Webpages</td>
<td>19</td>
<td>22</td>
</tr>
<tr>
<td>Number of Less Frequently Changed Webpages</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>Number of Not Modified Webpages</td>
<td>40</td>
<td>37</td>
</tr>
</tbody>
</table>

### 4.4.6 Discussion of Results

**Webpage Change Behavior:** For PDFC, on an average 19, 41 and 40 webpages were found changed in the category of “Frequently Changed”, “Less Frequently Changed” and “Not Modified” webpages respectively; while in PMCS 22, 41 and 37 webpages were found in the category of “Frequently Changed”, “Less Frequently Changed” and “Not Modified” webpages respectively, as shown in Table 4.1, and the same data is represented with the help of bar chart in Figure 4.7.

PDFC shows three webpages less than the PMCS in the category of “Frequently Changed” webpages. This difference between the numbers of webpages is due to
the techniques used for webpage change detection. PDFC considers the change in webpage only if the content of webpage is changed whereas in PMCS the webpage change is considered when the number of ‘HTML Tags’ were found changed even though the content of the webpages didn’t change. This change detection proves the proposed PDFC superior than PMCS.

![Figure 4.7: Average Number of Webpages Retrieved](image)

**Load on the Network:** The overhead caused by PDFC was 20KB (due to some additional load carried by the crawlers) which was constant for all the crawlers. The average load on the network caused by PDFC without Compression was 500KB which was computed by using the equation (4.1). The overhead in PMCS without compression was 30KB, whereas the average load was 534KB.

The difference between average load on the network caused by PDFC and PMCS was 34KB which shows that PDFC has reduced the average load on the network by 34KB which is considerable as compared to total size of data i.e. 800KB.

Load on the Network caused by PDFC with compression was computed using standard compression tool ‘WinZip’ and it was 164 KB whereas for PMCS with compression it was 176KB.
The difference between average load on the network caused by PDFC and PMCS was 12KB. The difference shows that PDFC has reduced the average load on the network by 12KB which is considerable.

The above calculated values are shown through a bar chart as shown in Figure 4.8 where the load on the network caused by different crawlers is taken along X-axis and the load in KB is taken along the Y-axis.

![Bar Chart](image)

Figure: 4.8 Load on the Network

The bar chart shows that the PDFC with or without compression performs much better than PMCS. Therefore, PDFC is able to save the load on the network due to its efficiency of targeting only those webpages which have been changed since the last crawl. This has improved the performance of PDFC as compared to the PMCS.

**Time Consumption:** On the basis of channel size of 4KHz and data transfer rate of 8KBPS, the time required to send the filtered webpages was calculated using equation (4.2). The average time taken by PDFC without compression to send the filtered webpages was 62.5 seconds whereas in PMCS without compression it was found 66.75 seconds. Thus, the difference between time consumption of
PDFC and PMCS shows that PDFC required lesser time as compared to PMCS to download the same set of 100 webpages. The difference is significant and hence efficiency of PDFC is improved.

The average time taken by PDFC with compression to send the filtered webpages was found 20.5 seconds whereas for PMCS it was 22 seconds. Thus, the difference between time consumption of PDFC and PMCS shows that PDFC required lesser time as compared to PMCS. The difference is significant and hence efficiency of PDFC is improved.

The results are also shown through a bar chart for better understanding as shown in Figure 4.9 where the load on the network caused by different crawlers is taken along X-axis and the time consumption is taken along the Y-axis. The bar chart shows that the PDFC with or without compression performs much better than PMCS.

As PDFC and PMCS provide better performance with compression, so the number of packet send to the RS and received by the SE was also recorded (by using the Activity Status of the computer on which experiment was performed). The achieved results are shown in Table 4.2.

The difference between total number of packets received and send by PDFC was 1060 in 9 seconds whereas for PMCS, it was 2960 in 34 seconds which was almost 3 times of PDFC (see equation (4.2) and Table 4.2). So in terms of number of packets sent to RS, received at SE and time required to crawl the webpages, the authors claim that PDFC is better than PMCS.
Network Utilization: The network utilization (discussed in section 4.4.4) was compared for PDFC and PMCS with compression by using Windows Task Manager as shown in Figure 4.10 and 4.11 respectively. The peak values of PDFC and PMCS were compared and it was found that in PMCS the peaks were high (see Figure 4.10) as compared to those in PDFC (see Figure 4.11). On comparison of peak values of PDFC and PMCS it was found that PDFC was more efficient in terms of network utilization as compared to PMCS.
Figure 4.11: PMCS Performance on Windows Task Manager

<table>
<thead>
<tr>
<th>Status</th>
<th>PDFC</th>
<th>PMCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Packets Sent</td>
<td>53212</td>
<td>42412</td>
</tr>
<tr>
<td>Number of Packets Received</td>
<td>54272</td>
<td>45372</td>
</tr>
<tr>
<td>Total Time taken to crawl 100 webpages</td>
<td>9 Seconds</td>
<td>34 seconds</td>
</tr>
</tbody>
</table>

4.5 Summary

In this chapter problems associated with the existing traditional and mobile crawler techniques were identified such as inefficient webpage change detection, consumption of memory and wastage of network resources. To address these problems the authors of the thesis proposed a novel architecture called Parallel Domain Focused Crawler for Reduction in Load on the Network (PDFC). The main features of PDFC are generation of MCs, allocation of URLs to MCs, detection of webpage change with minimum requirement of network resources, compression at RS and decompression at SE, construction and updation of data
in database, assignment of URLs in their respective domain queue and
distribution of load to different CHs.

The PDFC was implemented in Java using Netbeans IDE 7.0.1. The
performance of PDFC was evaluated and compared with the PMCS which was
the best crawler reported in (Nath and Bal 2011). They were compared on the
following four parameters (i) Webpage Change Behavior (ii) Load on the
Network (iii) Time Consumption and (iv) Network Utilization.

Experimentally, it was found that on an average 60% webpage changed and
hence only 60% of the total webpages need to be sent for indexing, thus
reducing 40% load on the network. This load was lesser than the load caused by
the PMCS. Furthermore, the comparison on peak values of PDFC and PMCS on
Windows Task Manager proves the proposed PDFC was found superior than the
PMCS. In terms of number of packets sent to RS and received by SE, the PDFC
transmitted approximately three times lesser number of packets. This proved that
the use of PDFC reduced the system overhead. Moreover, PDFC required less
time to crawl the same set of webpages as compared to PMCS. Thus the authors
have shown that the use of PDFC has reduced traffic on the underlying network
and has been found more efficient as compared to the PMCS.