CHAPTER 3

ARCHITECTURE OF MIGRATING PARALLEL WEB CRAWLER

3.1 Introduction

The Internet is a global system of interconnected computer networks. Due to the expanding nature of the web, it is a challenge to traverse all URLs in the web documents; it is important to parallelize a web crawling process. This chapter proposes a novel architecture of migrating parallel crawler, makes crawling task more effective and efficient among the different web crawlers which download web pages related to different domains. As the size of the web is very large, it becomes impossible to crawl the entire Web by a single process. Search engines run multiple processes in parallel which are referred as parallel crawler. While many search engines use a parallel web crawler, there has been little scientific research conducted on the parallelization of a crawler. Also as the number of crawling processes increases, the quality of downloaded pages becomes poor, unless they exchange back link messages.

The searching and indexing tasks for the web are currently handled from specialized web applications called search engines. The modern search engines can be divided into three components they are the publically available search engine, the database and the web crawling system. A web crawler is an automated program that browses the Web in a methodological manner. The process of traversing the web is called Web crawling. Web crawler starts with set of URLs to visit called as seed URLs. As it visits these pages, it scans them for links to other web pages. The Web Crawler consists of Crawler Manager, Robot.txt downloader, Spider and Link Extractor. A Crawl Manager takes a set of URLs from Link extractor and sends the next URLs to the DNS resolver to obtain its IP address. Robot.txt file are the means by which web authors express their wish as to which pages they want the crawler to avoid. Link extractor extract URLs from the links in the downloaded pages and sends the URLs to the crawler manager for downloading afterwards.
3.2 Problem Definition and Scope

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 which makes web crawling system more effective and efficient. The characteristics 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 another advantage 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. The crawler is implemented in Java. The implemented model supports all features of the real time III tier architecture. In this thesis a neural network based change detection method in migrating parallel web crawler is implemented. The method for Effective Migrating Parallel Web Crawling approach detects changes in the content and structure using neural network. Neural network based change detection method in migrating parallel web crawler will yield high quality pages and detect for changes, will always download fresh pages.

The crawling process is carried out using either of the following approaches: Crawlers can be generously allowed to communicate among themselves or they cannot be allowed to communicate among themselves at all. Both approaches put extra burden on network traffic. Here a fuzzy logic based algorithm is proposed and it is implemented in MATLAB using fuzzy logic tool box which predict the load at particular node and route of network traffic. The architecture of migrating parallel web crawler is validated using finite state machine. Test cases are generated for the validation of the architecture. The
approach for generating the test cases through FSM is very reliable and efficient and does not support for the invalid test cases. Valid input strings are generated as test cases. Empirical validation is also performed using SPSS tool to assess the performance of the proposed approach.

### 3.3 Proposed Architecture of Migrating Parallel Web Crawler

The figure 3.1 shown below is the proposed architecture of migrating parallel web crawler:

![Proposed Architecture of Migrating Parallel Web Crawler](image)

**Figure 3.1: Proposed Architecture of Migrating Parallel Web Crawler**
3.4 Algorithm

BEGIN: Crawler Process
LOOP(Till True):
REGISTER: Crawler process to Central Coordinator System
//Crawling Process wait for domain specific set of URLs to crawl//

INPUT:
KnownUrl=[url1 or url2 or …… urln]
//KnownUrl consists of domain specific set of URLs received from central coordinator system//
Where, url1,url2 ,……urln € D and D is specified Domain.

LOOP TILL: url!=Null
READ: url //from set knownUrl//
DOWNLOAD: Robot.txt
//Robot.txt carries downloaded information and also includes the
files to be excluded by crawler//
DETERMINE: Host Protocol
For Host Protocol: //Download the document //
CALL: multithreaded_downloader_process()
READ: Entire page and Remove the stop
word and Suffix
CHECK: Document downloaded or not
TRUE: Parse the page and find the url
hyperlink in the page
CHECK: url € knownUrl
TRUE:
CALL: Linkanalyzer()
//If document is fresh,
store the
Document, else abandon the
document//
ELSE EXIT:

END LOOP:
//Calculate page rank//

CALL: PageRank()
Compress and filter the downloaded document//
//Send the compressed and filtered content to central database of central coordinator
system//

CALL NNwed()
// Neural network method for web page change detection//
Initialize: input vector (v) values
Initialize: target vector (t) values
STEP1: Create neural network // Feed Forward Neural Network
STEP2: Training the neural network // Back propagation technique
STEP3: The result of training
STEP4: Simulating the neural network
STEP5: Analysis of result obtained
CALL Fuzzynetwrkldred() 
STEP1: Defining FIS variables and fuzzification of the input variables using membership function editor.
STEP2: Specifying rules for Fuzzy inference system using Rule Editor for network load reduction.
STEP3: Rule Evaluation
STEP4: Aggregation of the rule output
STEP5: Defuzzification of the output value

CALL Webcrawler_Validation() 
STEP1: Architecture is analyzed.
STEP2: UML Diagram of the defined Architecture is designed.
STEP3: The paths defined in UML are further graphically represented by the use of the concept of Finite State Machine (FSM).
//A finite state machine M is defined as: M = (Q, Σ, δ, q₀, F)
//A finite set of state in finite state machine denoted by Q.
//A finite set of input symbols denoted by Σ, these symbols may be any alphabets or numbers.
//A transition function is denoted by δ that takes as arguments a state and input symbol and returns a state. In the graph representation of an automata, δ is represented by an arcs between two states and label of arcs.
//If q is a state and a is an input symbol, then δ (q, a) is that state p such that there is an arc labeled a from q to p.
//q₀ is denoted as initial state, which is one of the state of Q.
//F is a set the final or accepting state, the set of F is a subset of Q.
STEP4: With the help of state diagram, a finite state machine is drawn according to UML state diagram and is represented in transition table.
STEP5: From the FSM, the transformation of states from one state to another state on the basis of events. The various productions can be induced for the finite state machine and the corresponding transition table is derived.
STEP 6: For verification of the production rules, some test cases are generated.

END LOOP:
END

3.5 Brief Description of Modules

In the first module, the architecture for Effective Migrating Parallel Web Crawling approach is proposed with domain specific and incremental crawling strategy that makes web crawling system more effective and efficient. 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. The web crawler as a client server system is implemented based on JAVA, 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 NetBeans 7.4 is used as Development Platform. In the second module, a neural network based change detection method in migrating parallel web crawler is implemented. The method for Effective Migrating Parallel Web Crawling approach detects changes in the content and structure using neural network. Neural network based change detection method in migrating parallel web crawler will yield high quality fresh pages and detect for changes. In the third module, discussion on the crawling process is carried out using either of the following approaches: Crawlers can be generously allowed to communicate among themselves or they cannot be allowed to communicate among themselves at all, both approaches put extra burden on network traffic. Here a fuzzy logic based algorithm is proposed and it is implemented in MATLAB using fuzzy logic tool box which predict the load at particular node and route of network traffic. In the fourth module, the architecture of migrating parallel web crawler is validated using finite state machine. Test cases are generated for the validation of the architecture. The approach for generating the test cases through Finite State Machine is very reliable and efficient and does not support for the invalid test cases. Valid input strings are generated as test cases. Empirical validation is performed to assess the performance of the proposed approach.

3.6 Conclusion
The size of the Web is very large and it is impossible to get the a perfect snapshot, led to the development of the architecture for Effective Migrating Parallel Web Crawling approach with domain specific and incremental crawling strategy, that makes web crawling system more effective and efficient. The implemented model supports all features of the real time III tier architecture. Many of the problems described in the previous chapters are avoidable with the right design. Several of issues were overcome by falling back on more simple technology. In this thesis a neural network based change detection method in migrating parallel web crawler is implemented. A fuzzy logic based algorithm is proposed and it is implemented in MATLAB using fuzzy logic tool box which predict the load at particular node and route of network traffic. The architecture of migrating parallel web crawler is validated using finite state machine. Test cases are generated for the validation of the architecture.