Fast Processing of Web Log Data Using Parallel Computing

By Hongzhi Li

A Thesis submitted to

the Faculty of Graduate Studies

In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

Department of Computer Science

University of Manitoba

Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

Fast Processing of Web Log Data Using Parallel Computing

BY

Hongzhi Li

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

Of

Master of Science

ABSTRACT

With the proliferation of websites and increased E-commerce systems, maintaining web log data has become very important for multitude of services. Web log data provides significant information about users behaviours which are used by companies to improve their websites and customized services. The insights from web log mining can help improve website design and personalization strategies. In web usage mining, about 80 % of the time is spent on data preprocessing. Parallel computing techniques can improve the performance and reduce the time taken for web usage mining. In this thesis, we propose to use parallel computing techniques to preprocess raw web log data and apply parallel association rule mining algorithms to extract user access-patterns from web log data. The results of the implementation are promising for future work in web usage mining.

ACKNOWLEDGEMENT

I would like to express sincere gratitude to my thesis supervisor Dr. Parimala Thulasiraman for her help and support in guiding me towards my goals. I am also grateful to the distinguished member of my thesis committee Dr. B. Guo, Dr. Rasit Eskicioglu and Dr. Peter Graham. I am also grateful to Dr. John Bate, Dr. Peter Graham for helping me with the administrative details regarding my thesis.

This thesis would not be successful without my family and friends who helped me at all time. Finally I thank to my wife Rong for her support my study.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

With the proliferation of websites and increase in E-commerce systems, maintaining web log data has become very important for multitude of services. Thousands of people may visit a web site in a given period of time. These visits are stored as web log data. Web log data provides significant information about users' behaviours which are used by companies to access their websites. The insights from web log mining can help improve website design and personalization strategies. Personalizing websites can attract new customers and retain existing customers. Web personalization can be accomplished by recommender systems [56], which provide recommendations to a user based on an analysis of previous users behaviours. Recommender systems can provide new information to match the needs of the individual web user [53]. Web personalization technologies also benefit E-commerce applications. It allows users to see and receive information based on the knowledge acquired of the users previous actions. For example, when a user logs on to a website to purchase bread, the website could recommend jam and ham to the user. The recommender system first needs to extract information from the characteristics of the web and user behaviour before making recommendations to the individual user. There are two kinds of data analysis techniques used on web personalization: *Collaborative filtering* and *data mining*.

Collaborative filtering [41, 55] is a popular data analysis method. It has been used in some successful E-commerce systems [32, 55]. The collaborative filtering systems are based on web users' rating of objects in a given domain. By finding similar preferences of users, collaborative filtering system can provide recommendations for users. The collaborative filtering techniques rely heavily on

web users' participation in order to provide recommendations for current users. For example, The GroupLens [55] research system provides recommendations on Usernet news and movies. The LikeMinds personalization server [32] makes product recommendations for a user based on the user's product preferences. Ringo [58], developed by MIT Media-lab, provides recommendations on music for users.

Most recently, data mining techniques are used to find useful information from web documents or web services [27]. Mobasher et al. [46] propose using web usage mining as an underlying approach for web personalization. Data mining techniques applied to the web personalization system can automatically analyze web usage data.

Web usage mining can be categorized into: 1) web data collection, 2) web data modeling (data preprocessing), 3) web data analysis, 4) web data recommendations provided by the personalization system [26]. In this thesis, we focus on data collection, preprocessing and using data mining algorithms to analyze web usage data. Recommendation systems for web usage are not discussed in this thesis.

1.1 Background

In the next three sections, we discuss data collection, data preprocessing, and data analysis.

1.1.1 Data Collection

Data collection is the first stage in a web personalization system. Web usage data can be collected from different sources. In general, the data can be collected from *web server side*, *proxy server*, and *user side (web browser).*

The data logged in server log files represent the access of a web site by multiple users. It explicitly records the browsing behaviours of web site users. A packet sniffer such as *tcpdump* [34] is an alternative approach to get the usage data from TCP/IP packets. The packet sniffer monitors the network traffic of website servers to capture the data. However, packet sniffers can not get data from websites that use secure socket layer (SSL) techniques [13]. In many financial or bank websites, the entire website is secure. Therefore, packet sniffers do not work at the secured websites.

A web proxy server works as an intermediate caching system between user browsers and website servers. When the proxy server receives a request for a web page from a user, the server first looks in its local cache of previously downloaded web pages. If the requested page is found, the proxy server returns the web page to the user without forwarding the request to the website. If the requested page is not in the local cache of the proxy server, the proxy server requests the page from the website server. Then the proxy server forwards the returning page to the user. The users do not realize the actions of the proxy server. Using the proxy server can reduce the loading time of the web page requested by the users [19]. Proxy log data provides the traffic from multiple users to multiple web servers.

We can also collect data from the user side. This requires users to enable the functionality of remote agents (such as JavaScript or Java applets) or modified browsers (such as Mosaic or NetZero) to capture the web usage data from the user side. The remote agents have a better performance on session identification problem than the server side data collection methods. However, the disadvantage of Java applet-based remote agents is that they consume some additional time when they are loaded for the first time. The JavaScript-based agents spend little time on interaction with web servers but cannot record all user clicks (such as back buttons). The user remote agents capture only single-user, single-site accessing actions. The modified browsers have data collection capabilities by modifying the original codes of existing

browsers. This method is versatile. However, convincing users to use this kind of browsers for visiting websites is a concerned problem. Table 1.1 shows the summary of the three types of data collections.

Data collection Types	Data Range
Server Side	Multiple Users to Single Site
Proxy Server Side	Multiple Users to Multiple Sites
User Side	Single Users to Multiple Sites

Table 1.1 Web Usage Data Collection Types

The primary data sources used in web usage mining are the web server log files. In this thesis, data is collected from web server side. We use web server log files as data sources of web usage mining research.

1.1.2 Data Preprocessing

Web log files are text files in which there is a record for each HTTP transaction. The following is an example of the information in a web log data:

123.456.78.9 - - [26/Feb/98:15:34:52 -0600] ``GET B.html HTTP/1.0'' A.html Mozilla/3.04 (win95, I)

Here, `123.456.78.9' is the IP address of the user (issuing HTTP requests). '- -' refers to the value "blank" which means the user is not authenticated by HTTP authentication. [26/Feb/98:15:34:52 -0600] is the time the user made the request. `GET' is the method of the transaction. `B.html' is the URL requested by the user. `HTTP/1.0' refers to the HTTP protocol. `A.html' is the referrer URL which is the URL the user accessed before B.html. `Mozilla/3.4 (win95, I)' indicates the user's

operating system and type of browser. In web log data, we are only interested in URLs requested by users. By mining the URLs in the web log data, we can discover user access-patterns in websites. Therefore, we can provide predictions for the users' possible future behaviour using recommender systems. Discovering user access-patterns in the web log data of websites requires a lot of computational power because the websites contain huge datasets (such as transactions). Before mining web log data, the web log data needs to be preprocessed. Actually data preprocessing is not a trivial task. 80% of the time in a knowledge discovery project is spent on preprocessing [42]. Reducing the preprocessing time can significantly improve the overall performance of the web usage mining task. Thus, ways of reducing the preprocessing time are important. In this thesis, we will use parallel computing techniques to preprocess raw web log data.

Data preprocessing is an important and necessary stage before using mining algorithms to analyze the web log data. Preprocessing of web logs includes four stages [20]: *data cleaning, user identification, session identification, and formatting data.* Data preprocessing will be discussed in detail in section 4.1.

1.1.3 Data Analysis

Web Log data analysis reveals the useful information from the log files of a web server. Many log analysis tools have been developed to analyze the data.

Log analysis tools, also called traffic analysis tools, use raw web log data as input to produce statistical information. The statistical information can provide the web site activity (such as the total number of visits, average time of view), diagnostic statistics (such as errors of page not found), server statistics (such as favorite pages accessed), clients statistics (such as user cookies, types of users' browser, users' operating system). There are several commercial web log analysis tools [2, 9], but

most of them are limited to simply producing frequency account. Therefore, log analysis is regarded as the simplest method to extract information from web log data. However, the log analysis tools only provide limited knowledge of users' behaviour [63]. Therefore, data mining techniques are becoming important.

Data mining [29] is a process of extracting certain hidden information from data. There are a variety of data mining techniques that have been applied to web usage data, including *clustering, classification, discovery of association rules* and *sequential pattern discovery.*

Clustering is a technique for grouping together objects that have similar characteristics. In the web domain, there are two types of clusters: user clusters and page clusters. Clustering of users provides groups of users having similar browsing behaviours when navigating through a web site. Clustering of pages establishes a group of web pages with related contents. Clustering can be used to extract user information and data from web log files. In the context of web usage mining, Shahabi et al. [57] and Joshi et al. [36] used clustering techniques to discover clusters of web users having similar information. The knowledge of the clusters of web users can be used to perform market segmentation in E-commerce systems.

Classification is a process of assigning a data item into a set of predefined classes. In the web domain, the classes could be different user profiles. A user profile describes a group of users with common attributes and may include demographic information about users (such as name, country, age, or sex), user's interests or behaviours of users accessing websites. Therefore, classification in web usage mining may discover demographic information, such as "50% of the clients who visited the web page of product2 were from the East Coast" [21].

Association-rule mining (*ARM*) can explore the potential relationships among items in datasets. ARM finds frequent itemsets and association rules among sets of items. An association rule has the form $A \Rightarrow B$ *(s, c),* where A and B are itemsets, s is

an expression of the support of the rule, and c is the confidence of the rule. The support s is a ratio of the number of transactions that contain both A and B itemsets to the total number of transactions. The confidence c is a ratio of the number of transactions that contain both A and B items to the number of transactions that only contain item A. An association rule can be generated when its support and confidence are greater than minimum thresholds, respectively. In this research, an item is a URL and each transaction is a set of visited URLs.

Association-rule mining algorithms operate in two stages:

- Finding all frequent itemsets whose supports satisfy a minimum threshold, and
- Extracting rules whose confidence values satisfy a minimum threshold.

Apriori [12] is a well-known association rule algorithm. Apriori performs a number of passes to find frequent itemsets. Each pass has two phases:

- Computing supports to find frequent items, and
- Generating candidates for the next pass.

During the L^{th} pass, Apriori finds the set of frequent itemsets having L items.

In web usage mining, ARM techniques can reveal correlations among web pages accessed by users. Fu et al. [30] and Mobasher et al. [46] propose using association rule techniques to automatically extract information from web log data for making recommendations. Pramudiono et al. [51] utilize association rule techniques to discover mobile user profiles for optimizing website services provided to them.

Sequential pattern discovery is an extension of association rule mining that finds patterns consisting of the presence of *a set of* items followed by another *set of* items incorporating the notion of transaction time sequence. Buchner et al. [17] propose the MiDAS data mining algorithm to reveal sequential patterns from web log files. Sequential patterns can be used to predict future users' visiting patterns.

1.2 Goal of the thesis

In web usage mining, about 80 % of the time is spent on data preprocessing. Data preprocessing is one of the first steps that needs to be accomplished before web data analysis. In this thesis, we propose to:

- use parallel computing techniques to preprocess raw web log data, and
- apply parallel association rule mining algorithms to extract user access-patterns from web log data.

Applying these proposed techniques on raw web log data could significantly reduce the time taken for web usage mining.

1.3 Organization of Thesis

The organization of the rest of this thesis is as follows:

- *Chapter 2: Web Usage Mining.* This chapter describes web log data preprocessing and user pattern discovery stages.
- *Chapter 3: Parallel Computing.* This chapter briefly presents computing concepts.
- *Chapter 4: Parallel algorithms design.* This chapter describes both sequential and parallel data preprocessing and association rule mining algorithms..
- *Chapter 5: Experimental Results.* This chapter describes performance analysis of the parallel algorithms.
- *Chapter 6: Discussion and Conclusions.* This chapter gives a summary of the thesis and future work.

Chapter 2

Web Usage Mining:

Background & Related Work

Web usage mining uses data mining techniques to extract web user access patterns from web log data to predict users' behaviours. In this chapter we briefly discuss data preprocessing and association rule mining.

2.1 Web Usage Mining Architecture

Mobasher et al. [46] propose using association rule mining to automatically extract important information from web log data for personalizing websites (called *usage-based personalization*). Figure 2.1 is the general architecture of usage-based personalization system. Usage-based personalization system includes three components: data preprocessing (block A), data analysis (block B), and recommender engine system (block C).

In this thesis, we only concentrate on blocks A and B. The decomposition of blocks A and B is shown in Figure 2.2. In the next few subsections, each of the components of Figure 2.2 is discussed.

2.2 Web Log Data Preprocessing

A web log file contains information about users' accesses to a website. Before using data mining techniques to discover users' access patterns from web log files, we need to preprocess the raw web log data. We will provide brief descriptions of several

abstractions concerning web usage data, before discussing web log data preprocessing.

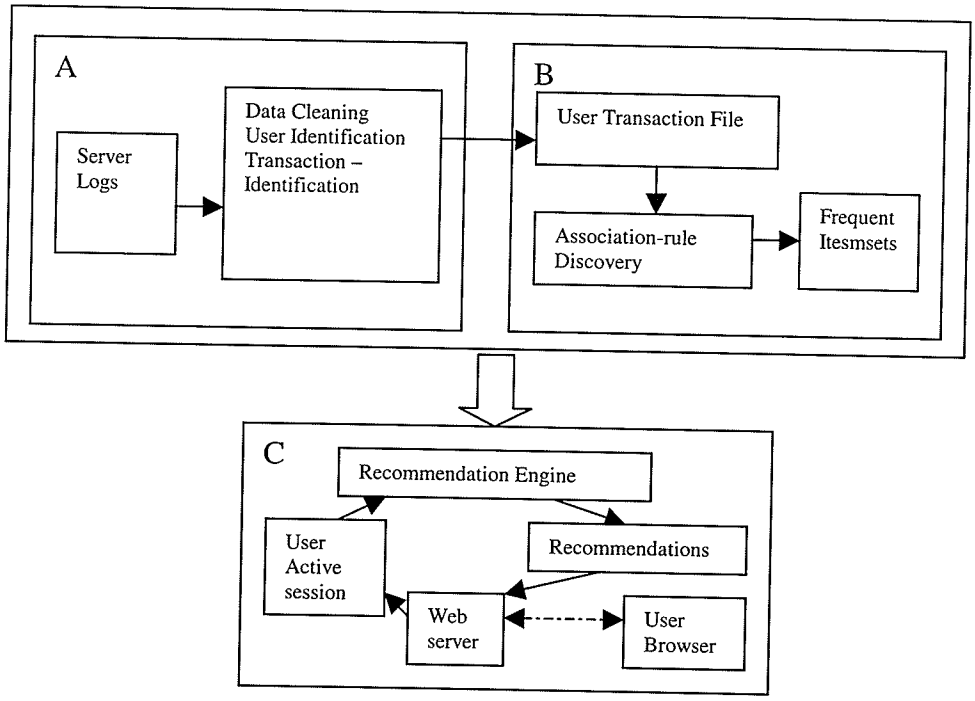

Figure 2.1. Web Usage-Based Personalization

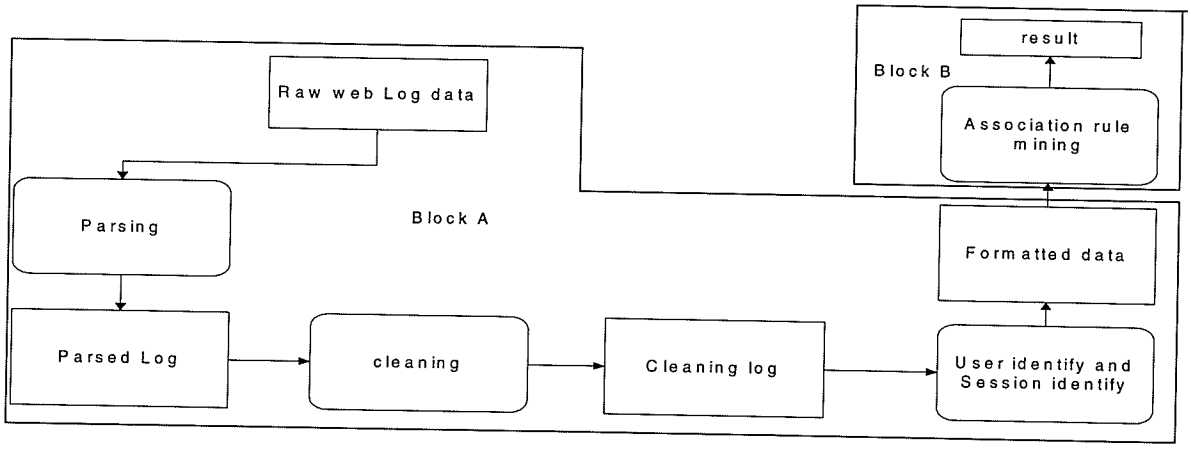

Figure 2.2. Web Log Data Preprocessing and Web Log Mining

The W3C (World Wide Web Consortium) Web Characterization Activity team [43] has published a draft of web term definitions such as *users*, *page views*, *user sessions*, and *server sessions*. According to their definitions, a user is an individual who is using a browser to access one or more websites. A page view is a visual rendering of a web page that is displayed on the user's browser at some time. A user session is a set of page views that the user accesses across the entire web. A server session is a collection of page views belonging to a single website accessed / used during a user session.

Most web servers provide functions to store log files. The log file stores every user's access to a web page on the web site. The entries of log file consist of several fields that are predefined by web server. The common log file format [4] includes the following fields:

remotehost rfc931 authuser [date] "request" status bytes

where ***remotehost*** refers to remote hostname or IP address if DNS hostname is not available, ***rfc931*** is remote logname of the user, ***authuser*** is a user name if the web server is using local authentication and registration, ***[date]*** refers to the date and time of the user's requests, ***"request"*** includes request's methods (such as GET, PUT, POST, or HEAD), file retrieval protocol, ***status*** refers to status code returned to the user, ***bytes*** is the content-length of file retrieved. The common log file format can be used by most web servers. An example of a common log record is:

123.456.78.9 - - [26/Feb/98:15:34:52 -0600] ``GET B.html HTTP/1.0'' 200 308

In many cases, web owners or web developers want to record more information. Therefore, W3C uses an extended log file format to collect more data for demographic analysis. The following is an example record of extended log data.

123.456.78.9 - - [26/Feb/98:15:34:52 -0600] ``GET B.html HTTP/1.0'' 200 308 A.html Mozilla/3.04(win95, I) MmTaUg00pdA00001fvkwsM4000

The main extension to the common log format results by adding few fields into the log file. The most important fields are: ***referrer*** *(e.g., A.html),* which is the URL the user accesses, ***user_agent*** *(e.g., Mozilla/3.04(win95, I))* indicates the user's operating system and type of browser, and ***cookie*** *(e.g., MmTaUg00pdA00001fvkwsM4000)* is a unique number sent to user's machine by the web server. Because the extended log file format data provides more information than the common log file format data, in this thesis, we use the extended log format files as input data.

If L is a set of log entries, then a log entry $l \in L$ has the following fields:

- the IP address of the user, denoted *l.ip;*
- the time of user accessing, denoted *l.time;*
- the URL of the web page visited by user , denoted *l.url;*
- the agent of user's operating system and type of browser, denoted *l.agent;*
- the cookie of user getting from the web server, *denoted l.cookie* .

Preprocessing of web logs includes four stages [20]: data cleaning, user identification, session identification, and data formatting. We will briefly discuss each of them in the following sections.

In most web log data mining research, researchers have focused on access-pattern discovery phase. They used different data mining techniques to extract users' access patterns. Although data preprocessing is a necessary phase in a knowledge discovery research, people seldom pay appropriate attention to data preprocessing. Researchers [42, 49] report that the actual data mining phase occupies at most 20% of the efforts in a knowledge discovery research, and about 80% of the time is spent on

data preprocessing. When data preprocessing time is reduced, the 80% of the time spent will also be decreased leading to faster web log mining.

Ansari et al. [13] propose an architecture to collect users' data from web application server "layer" rather than data from web log files. Using this architecture, web usage mining not only collects users' transaction data but also web content data (e.g., images, articles, and multimedia). Application servers can use cookies to trace a user's access behaviours. Associated with login mechanisms, web server can easily identify a particular visitor. The application server can be designed to keep track of users' behaviours even without cookies because the entire logical transactions of web server are implemented through the application server. Furthermore, the time of data preprocessing can be reduced through an appropriate data transformation mechanism from the web transaction processing system to a database or a data warehouse. This architecture has been made available to clients. The system has been purchased for E-commerce by some retailers such as Mountain Equipment Co-op, Bluefly, Canadian Tire, and others [38]. However, if existing web sites need to collect data at the application layer, redesign of the architecture would be necessary. The cost of such redesign of the architecture might be high.

2.2.1 Data Cleaning

The goal of data cleaning is to remove errors (such as accessing a non- exiting web) and irrelevant items (such as graphics) from the log data depending on the application. "HTTP (Hypertext Transfer Protocol) is an application-level protocol over TCP (Transfer Control Protocol) for distributed, collaborative hypermedia information systems [6]". When a user connects to the web server, the user can download text, graphics, and other elements comprising the web page. If web pages consist only of HTML, downloading HTML can be completed in a single connection.

Actually in most cases, web pages contain many elements such as text, graphics, audio, streaming video, and so on. Therefore, retrieving each elements of a web page requires a separate connection to the web server. When the user accesses web pages to download a HTML file, several entries are recorded in the web log file. These entries may include graphics, streaming video, or scripts in addition to HTML file. For instance, the home page of Department of Computer Science of the University of Manitoba (www.cs.umanitoba.ca/newsite/ index.htm) is shown in Figure 2.3 without graphics. In Figure 2.4, there are picture elements (arrow pointing) associated with the index.htm. When a user downloads this file, these picture elements are recorded in the log file.

In this thesis, we are only interested in the actions of the user interaction with the website. We are concerned with only HTML files. Therefore, all other irrelevant log entries (e.g., graphics files accessed) can be eliminated from the log data during cleaning. This process of eliminating irrelevant information from a log file is called data cleaning.

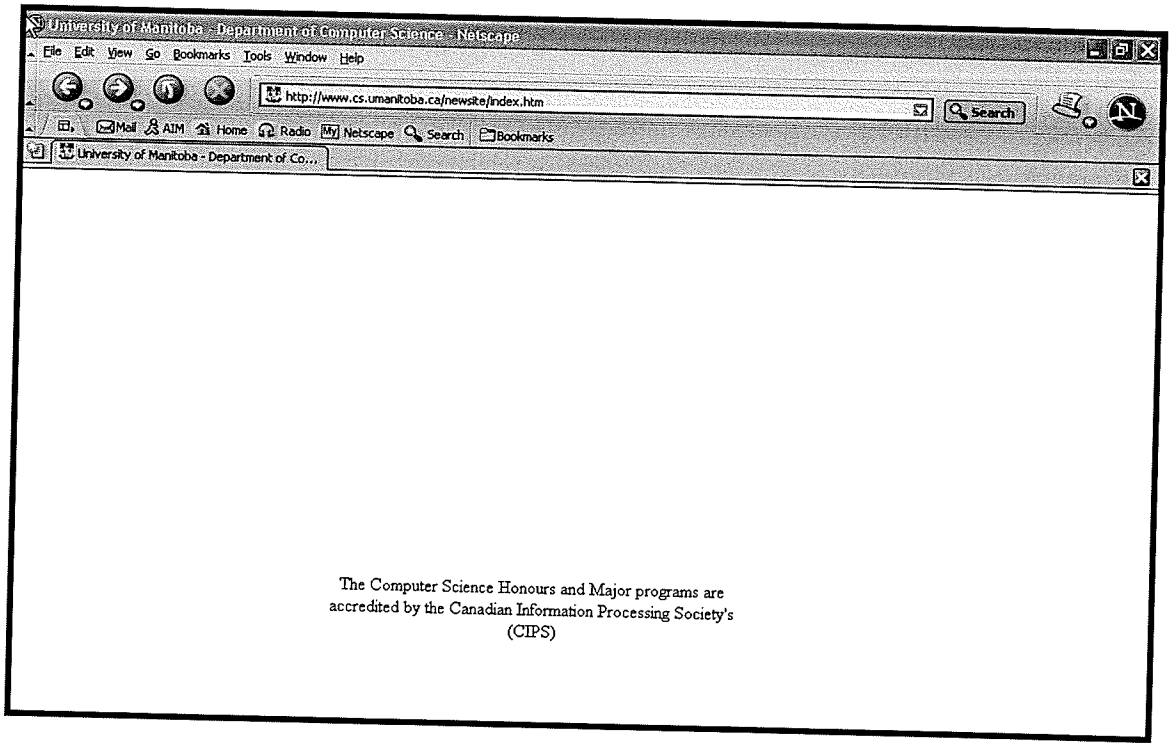

Figure 2.3. Example HTML Page

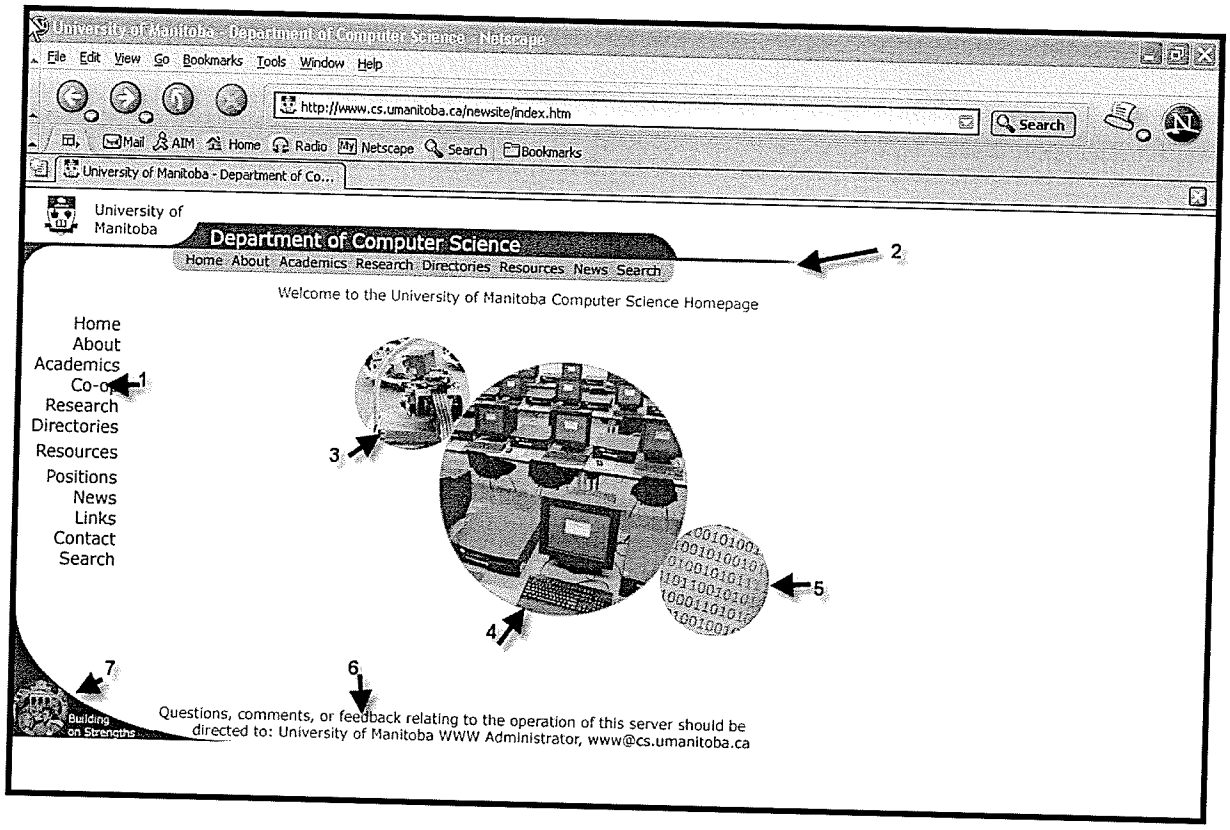

Figure 2.4. Associated Images

Elimination of irrelevant log entries can be implemented by checking the suffix of URL name. For example, all entries of log data including URL name suffix such as gif, Gif, jpg, JPG and image can be deleted. In addition, some other elements such as multimedia files (e.g., ".wav" or ".swf") can be removed. In this thesis, we use a default suffix list to eliminate irrelevant entries. However, every web site can modify the suffix list depending on its type. For instance, when a web site focuses on distributing graphics, analysts can not remove all of the GIF or JPG from web log data. The graphical files are important elements presenting users' actions. Therefore, a list of actual file name to remove can be defined by each website instead of the suffix list.

In recent years, more and more researchers pay attention to the impact of web traffic created by bots [35] and crawlers. Web robots, spiders, and crawlers are collectively called as bots. Bots and crawlers are software programs or agents that

automatically retrieve information from internet by traversing the hypertext structure of World Wide Web.

Bots includes search engines, web monitoring software, shopping bots, and e-mail harvesters. Internet search engines such as Google [5] and Altavista [3] build their index database depending on documents retrieved by web robots. Keynote [7] provides web sites' performance by monitoring and testing web-based applications. mySimon [8] is a shopping bot. mySimon provides product prices for users by scanning commerce websites. E-mail harvesters such as EmailSiphon and Cherry Picker try to look for e-mails to add into their sold e-mail lists. Bots increase entries in web log files.

To exclude robots, Kollar et al. [39] propose a Robot Exclusion Standard. According to this standard, web administrators specify off-limits parts of their websites in a robot.txt. Whenever a robot accesses a web site, it should first visit robot.txt file in the web root. Some parts of websites in the robot.txt cannot be visited by the robot. For instance, the following robot.txt

 # robot.txt for http://www.abc123.com/

```
User-agent:*
Disallow:/temp/
Disallow:/foo.html
```

specifies that no robots should visit any URL starting with "/temp/" or /foo.html.

However, many robots do not follow this standard. Tan et al. [61] and Kohavi et al. [39] discover that 5% to 40% of sessions are generated from web robots visits. If bots sessions are not correctly identified and removed, the web usage mining will make a wrong analysis. Therefore, the identification of bots and crawlers are very important in the web usage mining project.

The simplest method is to identify and remove bots session based on bots' user agent or IP (two fields) in the web log data. Various web robot repositories exist on the internet. The web robot repositories maintain a list of IP addresses or user agent of

most popular web robots. The well-known web robot repository is *www.robotstxt.org* that provides a database of common web robots. Table 2.1 lists some web robots. The bots list must be updated regularly.

Web robot name	IP address	HTTP User agent
Googlebot	64.233.167.99	Googlebot/2.X (+http://www.googlebot.com/bot.html)
Atomz.com Search Robot	64.191.196.98	Atomz/1.0
Email Harvester	4.41.77.204	EmailSiphon
Netscape(browser)	160.90.178.152	Mozilla/4.7[en](x11;I;Linux 2.2,14-5.0 i686)

Table 2.1. A Sample List of IP Addresses and User Agents

Web administrators or web designers always need to maintain or edit websites. Their accesses also are recorded in the web log files. These records are meaningless for web usage mining because web mining focus on users' behaviours. Therefore, eliminating local access records is a necessary task. In this thesis, we can identify and remove local access entries from web log data depending on IP address (e.g., 127.0.0.0).

In this thesis, we have developed an improved algorithm called *WebLogCls* for data cleaning based on a previous work [20]. An implementation of this improved algorithm is described in Chapter 4.1.1.

At the end of data cleaning process, we can get a temporary file *T*. The *T* is a set of entry. Each entry consists of *url, time, agent, IP address*, or *cookie* components. Other fields such as *rfc931*, *authuser*, *status*, *bytes*, and *refer* in the web log data are striped out.

2.2.2 User Identification

User identification process aims at trying to obtain information about the users accessing the particular website. In web log files, there are several fields that can be used to identify users. We can identify users by using the ***authuser*** in web log data. ***authuser*** is populated by the local authentication and registration mechanism. Mobasher [47] et al. applied the method of combining IP address and ***authuser*** to identify web users in the WEBMINIER [47] project. When a user logs into a website, ***authuser*** will be filled by a value. Otherwise, the ***authuser*** is a "-". Only registered users can be identified by ***authuser***. However, most of users accessing a website are un-registered clients. Due to this limitation imposed ***authuser*** in using identifying users, most web usage mining projects and log analysis tools use cookies or IP and user agent to identify users.

In Spilipoulou [59], experiments show that cookie-based user identifiers can increase the quality of session identification by up to 20%.

When log files do not include cookies, the simple heuristics proposed by Colley et al. [20] can be used to implement user identification. For example, Table 2.2 shows a sample of web log data. There are no users' IDs in the web log file. Although the IP addresses are the same, the browser agents are different. It can therefore be inferred that there are at least two user sessions on the machine. Based on agent information, it can be surmised that pages A, B and C are accessed by one user and pages L and R are visited by the other user. Hence, using such heuristic methods, two users are identified with browsing paths of A-B-C and L-R.

No matter what kind of methods are applied to identify users, this task is difficult because of various levels of caching [50] used in web access. For example, browser caching can improve the apparent performance of websites. When a user returns to the previous page by clicking the "back" button, the cached page will be

displayed on the user's browser. However, the web server log cannot record the repeated page access. Another caching level is at the proxy server. Proxy caches are shared by many users. The web log only records one page request from the proxy server even if the request is repeated multiple times from many users.

In this research, we use IP and user agents to identify users. As mentioned, cookies can be used to implement user identification. When a user enters a web site, the web server can sent a unique number (cookie) to the user's browser which stores the cookie on the client's machine. On later accesses, cookies can be retrieved and by using cookies, web servers can trace a user even with a dynamic IP-address through more than one visit. A potential problem with using cookies is privacy issue. Web providers want to know who is using their services. Web users want to preserve their privacy. These goals are in conflict and some users disable cookies while others choose to allow cookies to be able to access websites that depend on them. Many websites (e.g., Yahoo, Amazon, and eBay) use cookies to trace users accessing behaviours.

2.2.3 Session Identification

A user may access the website more than once. The web log file stores the records of all the accessing periods. Therefore, many page views may belong to a user. At the end of user identification section, a necessary method has to be used to divide page views of each user into individual sessions. One simple method to approach this problem is to set a timeout. When websites are developed, designers use session objects (e.g., ASP's session and Java's Httpsession) to implement developments. Web designers need to set a time value to session objects in order to make session object invalidate if users do not do anything after this time. This value is always set to 30 minutes.

IP Address	User ID	Time	Method/URL/Protocol	Status	Size	Referrer	Agent
123.456.78.9	-	25/Apr/98:03:04:41	Get A.html HTTP/1.0	200	3290	-	Mozilla/3.04 (win95, I)
123.456.78.9	-	25/Apr/98:03:05:34	Get B.html HTTP/1.0	200	2050	A.html	Mozilla/3.04 (win95, I)
123.456.78.9	-	25/Apr/98:03:05:39	Get L.html HTTP/1.0	200	1389	-	Mozilla/3.01(X11, 1, IRIX6.2, IP22)
123.456.78.9	-	25/Apr/98:03:07:55	Get C.html HTTP/1.0	200	8289	B.html	Mozilla/3.04 (win95, I)
123.456.78.9	-	25/Apr/98:03:09:50	Get R.html HTTP/1.0	200	1680	L.html	Mozilla/3.01(X11, 1, IRIX6.2, IP22)

Table 2.2. A Sample of Web Log data

Madria et al. [45] propose the thirty-minute timeout based on this empirical data. Therefore, in this thesis we use a thirty-minute timeout to bind each session of a user. Figure 2.5 illustrates an example information of a user's page views. The user visits *A, B, C, D, C, B, A, B* web pages within a certain period of time. Based on 30 minutes time out, we can bind 3 sessions of this user.

In this thesis, we have developed an improved algorithm called *WebLogIde* for user identification and session identification based on a previous work [20]. An implementation of this improved algorithm is described in Chapter 4.2.1.

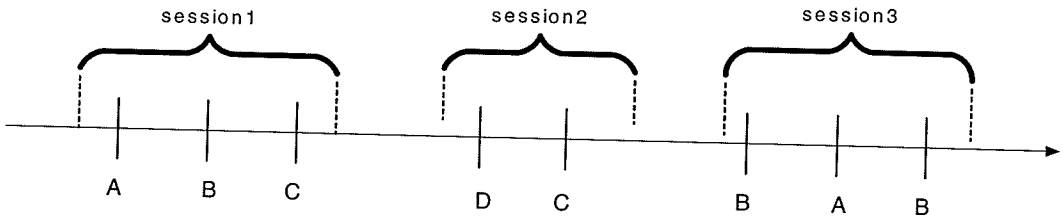

Figure 2.5. Session Identification

2.2.4 Data Formatting

Finally, the web log data needs to be formatted for association rule mining applications. Since the goal of this research is to extract users' access-patterns from web log data, irrelevant attributes (e.g., time, IP and user agent) of each item (URLs in this research) should be striped out. In this thesis project, this step is very simple, because we only output URLs into a text file and the time field is not written into the text file.

2.3 Web User Pattern Discovery

Discovering user access-patterns from web log data can reveal useful and interesting information. The knowledge discovered can assist websites to improve their services

and provide recommendations to web users. There are several algorithms that have been applied for mining web log data, such as clustering [36, 57], classification [21, 28], association rule mining [10, 12, 30, 46, 51], and sequential pattern discovery [17]. In this thesis, we focus on using association rule mining techniques to discover the potential relationship among user access behaviours.

Kitsuregawa et al. [37] and Demiriz et al. [25] utilize parallel techniques in access-pattern discovery processing to improve the performance of user access-pattern discovery. Kitsuregawa et al. propose parallelizing SQL execution to implement parallel association-rule mining to extract information in web log data. In [37], web usage data come from the web log data of Mobile Info Search (MIS)[http://www.kokono.net]. The web log file is first cleaned and transformed to transaction table. The entire transaction table is stored in a parallel database system (called DBKernel) developed by Tamura et al. [60]. The proposed parallel SQL execution is based on DBKernel processing system. If other websites want to use this method, the websites must have the same parallel database processing system.

Demiriz and Zaki [25] propose webSPADE, which is a parallel sequence mining algorithm to analyze web log files. webSPADE uses the web log files from Verizon.com website. The web log data are first parsed based on the log data formats used by Verizon.com. Then the data are cleaned and user sessions identified using cookies. A sophisticated data warehouse is designed to store user session information and other operational data. The webSPADE algorithm is implemented using multithreaded programming techniques. This parallel sequence mining algorithm is based on a shared memory system. webSPADE can provide prediction of a user's likely future access pages and has been successfully applied by Verizon.com website.

Note that the work in the literature in parallel computing has mostly been concentrated on the data mining algorithms with very little emphasis on data preprocessing.

2.3.1 Association Rule Mining

Association rules mining (ARM) is concerned with mining of associations over sales data. In [10], Agrawal applied ARM to basket data to identify associations between items that are purchased together. Basket data present the records of database relating to supermarket customer transactions with each transaction storing items purchased by a customer during a visit. In ARM, each record in the database is referred to as a transaction consisting of attributes which are referred to as items. Given a large database of customer transactions, association rule mining focuses on discovering all significant associations between items in the database.

Given a set of transactions, where each transaction is a set of items, an association rule is expressed as $X \Rightarrow Y$, where X and Y are sets of items. This implies that if X occurs, then Y will also be present. X and Y are respectively called the *antecedent* and the *consequence* of the rule. This can be described as follows :

Let $I = \{i_1, i_2, \ldots i_m\}$ be a set of m distinct items.

Let $D = \{T_1, T_2, \ldots T_n\}$ be a set of variable length transactions, where each $T_i \in D$ contains a set of items $i_i, i_j, \ldots, j_k \subseteq I$. In a association rule $X \Rightarrow Y$, where $X \subseteq I$ and $Y \subseteq I$, $X \neq 0$, $Y \neq 0$, and $X \cap Y = 0$.

In general, an itemset consists of a set of items (such as the *antecedent* or the *consequence* of the rule). If the number of items in an itemset (such as X) is k, then X is called as a k-itemset.

For any given set of transactions, all association rules generated should satisfy two measures: *confidence* and *support*. The support is percentage of transaction that contains both X and Y itemsets among all transactions. The confidence is a ratio of the number of transactions that contain both X and Y itemsets to the number of transactions that only contain itemsets X. Support is a measure of significance of the

correlation between itemsets. Confidence is a measure of strength of correlation between itemsets. Therefore, the problem of mining association rules can be decomposed into two subtasks:

1. Discover all itemsets such that the supports are bigger than user specified minimum support (MS).
2. Generate all rules such that the confidences are above user specified minimum confidence (MC).

For example: a rule, $X \Rightarrow Y$, is generated when the support

$$\sup(X \Rightarrow Y) = \frac{number\ of\ transactions\ containing\ X \cup Y}{total\ number\ of\ transactions} \geq MS \qquad (1)$$

and the confidence,

$$conf(X \Rightarrow Y) = \frac{\sup(X \Rightarrow Y)}{\sup(X)} \geq MC \qquad (2)$$

The first task is to search the whole database to find all itemsets whose support is above user-defined minimum support. Such itemsets are called as *large itemsets* or *frequent itemsets*. Finding *frequent itemsets* can be divided into two subtasks: *candidates itemsets* generation and computing supports to generate *frequent itemsets*. The itemsets that are expected to be frequent itemsets are called as *candidate itemsets*. The process of discovering all large itemsets and their supports is computationally expensive. For instance, given m items, there are potential 2^m itemsets in the lattice [12]. Figure 2.6 shows the lattice for a four itemsets example. If the number of items is large, the lattice grows exponentially. However, only a small fraction of this large number of itemsets has minimum support. Thus, it is not necessary to search for the support of every itemset.

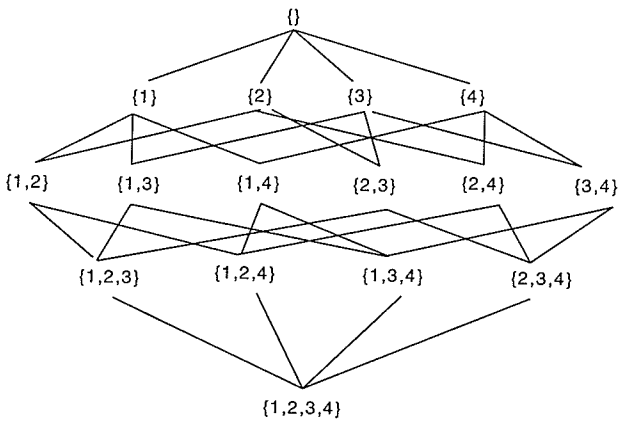

Figure 2.6. Lattice for a 4-itemset (c.f.[64])

The second task that generates all rules from given large itemsets is relatively straightforward. Therefore, developing an efficient method to find all potential large itemsets is the major concern of the association rule mining research.

2.3.2 The Basic Association Rule Algorithm (AIS)

Agrawal, Imielinski, and Swami (AIS) [10] algorithm is the first association rule mining algorithm proposed in the literature. The AIS scans a database for many passes to find frequent itemsets. For example, given a sample database (Table 2.3), during the first pass over the database, the AIS calculates the support count of each individual item as shown in Table 2.3 (b). Items I_1 and I_2 are found 7 times in the database; I_3 and I_5 are found five times and I_4 and I_6 are found two times. If we assume the *minimum support* is 30%, then by using equation 1 we get $sup(I_1) = sup(I_2) = 70\%$; $sup(I_3) = sup(I_5) = 50\%$. Since this is larger than 30%, these frequent itemsets are generated and shown in Table 2.3 (c). The items (I_4 and I_6) are eliminated from the frequent itemsets because these items' (I_4 and I_6) supports are less than 30%. Using the frequent itemsets generated, candidate 2-itemsets can be produced by

extending those frequent itemsets in the same transaction. The candidate 2-itemsets are generated in Table 2.3 (d) against assuming minimum support of 30%. During the second pass over the database, the support count of all candidate 2-itemsets are calculated as shown in Table 2.3 (d). Then, frequent 2-itemsets are generated against minimum support (Table 2.3 (e)). Similarly frequent itemsets can be extended to produce candidate itemsets for the next pass for 3-itemsets and so on. This process will terminate when one of frequent itemsets and candidate itemsets becomes empty. In this example, the result frequent itemsets include only one frequent 3-itemsets.

The main drawback of AIS is that it generates too many candidate itemsets that turn out be small [12]. For example, in Table 2.3 (c) item I_4 has been eliminated from frequent itemsets since its support is less 30%. Therefore, I_4 should not be considered as an element of a candidate in Table 2.3 (d). The other problem is that AIS requires too many passes over the entire database to discover frequent itemsets. Agrawal et al. [12] propose a modified version of ARM (called Apriori) to solve these problems. In this thesis, we used Apriori to generate frequent itemsets. The Apriori algorithm will be explained in detail in Chapter 4.

TID	List of Items
1	I_1, I_2, I_5
2	I_2, I_3
3	I_1, I_3
4	I_1, I_2, I_4
5	I_1, I_3, I_5
6	I_2, I_4, I_5
7	I_1, I_2, I_3, I_5
8	I_2, I_3, I_6
9	I_1
10	I_1, I_2, I_5, I_6

(a)

Items	Support
I_1	7
I_2	7
I_3	5
I_4	2
I_5	5
I_6	2

(b)

Frequent Itemsets
I_1
I_2
I_3
I_5

(c)

Items	Support
I_1, I_2	4
I_1, I_5	4
I_2, I_5	4
I_2, I_3	3
I_1, I_3	3
I_1, I_4	1

(d)

Frequent Itemsets
I_1, I_2
I_1, I_5
I_2, I_5
I_1, I_3
I_2, I_3

(e)

Items	Support
I_1, I_2, I_5	3
I_1, I_2, I_4	1
I_1, I_3, I_5	2
I_2, I_4, I_5	1
...	...

(f)

Table 2.3 AIS Mining Process

Chapter 3

Parallel Computing

Parallel computing provides a solution for large computational problems using more than one processing unit. Many problems [65] such as quantum chemistry, astrophysics, medical imaging have applied parallel computing techniques to achieve high performance computing. There are several ingredients necessary for parallel computing: operating system, programming model, parallel compilers and parallel algorithms.

In this chapter, we will discuss briefly the classification of parallel computers and parallel programming models.

3.1 Classification of Parallel Computers

Most large parallel commercial machines are classified based on Flynn's taxonomy of computer architecture [31]. There are four possible classifications depending on the parallelism exhibited in instruction and data streams: Single Instruction Single Data Stream (SISD), Multiple Instruction Single Data Stream (MISD), Single Instruction Multiple Data Stream (SIMD) and Multiple Instruction Multiple Data Stream (MIMD).

3.1.1 Single Instruction Single Data (SISD) Model

The SISD computer consists of a main memory and a processor. This is nothing but a sequential von Neumann machine. In this computational model, the processor executes instruction streams in sequence that operates on a sequence of data. Data and

instruction streams are transferred between the memory and the processor. An algorithm running on a SISD computer is termed a sequential algorithm. Modern superscalar architectures provide some concurrency of program execution for uniprocessor computers. Superscalar architectures dynamically retrieve and delegate multiple independent operations that might be executed simultaneously. Although the superscalar architectures provide some advantages, they still have the drawback that the performance of parallel programs cannot be improved significantly due to data communication bottleneck between memory and the processor, and vice versa.

3.1.2 Multiple Instruction Single Data (MISD) Model

A MISD computing system is a multiprocessor machine (shown in Figure 3.1). The processors may execute different instructions at different instance of time, but they all operate on the same data stream. Systolic arrays are MISD computing architectures. Systolic arrays are widely used in image processing applications due to the pipelined structure of the algorithms. Computers built using the MISD model, are not useful in most application systems; none of them are used in commercial environment.

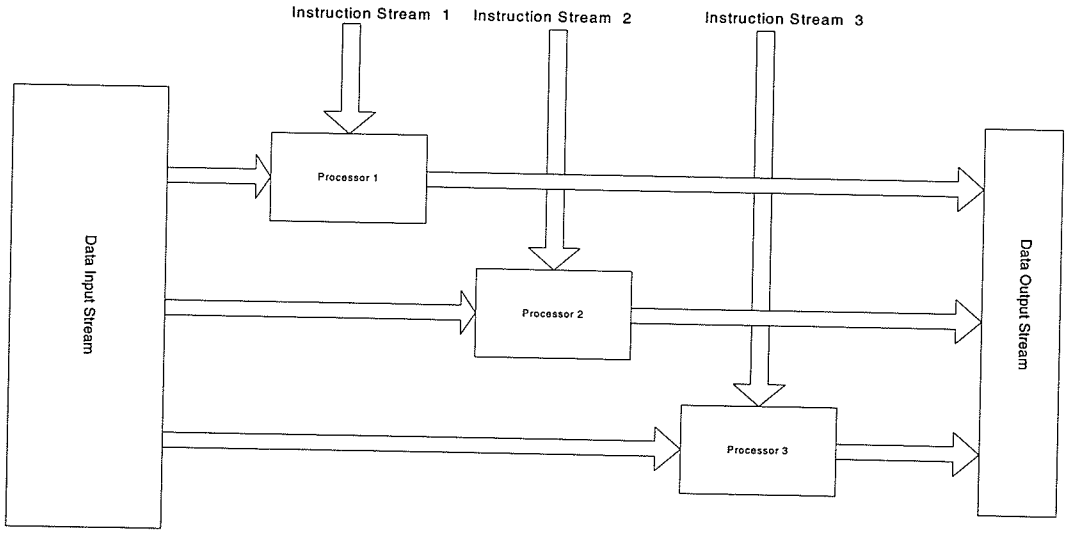

Figure 3.1 Multiple Instruction Single Data (MISD) Model

3.1.3 Single Instruction Multiple Data (SIMD) Model

SIMD architecture (Figure 3.2) consists of a linear array of processors that execute a single instruction on multiple sets of data in parallel. In SIMD model, a common global control unit (CU) provides a single instruction to all processing elements (PE). During each instruction cycle, all processing elements receive the same instruction from the CU. Each PE however, may execute the instruction on its own datasets. Therefore, at a given instance, all PEs operate synchronously by executing the same instructions on different data sets stored in their local memory. This kind of parallel machines are called synchronous programming models. The CRAY's vector processing machine and Thinking Machines's cm* are dominant representatives of SIMD architectures.

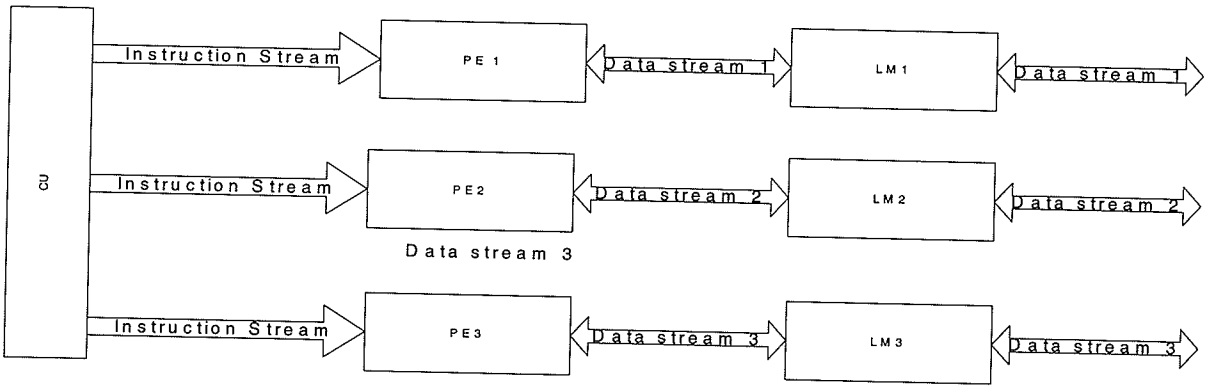

Figure 3.2 Single Instruction Multiple Data Model

3.1.4 Multiple Instruction Multiple Data (MIMD) Model

MIMD machines are mostly in use in many types of parallel machines. In a MIMD model, each of the PE has its own control unit and local memory. The PEs communicate with each other via an interconnection network. Each processing

element has its own set of instructions executed simultaneously with different data streams. Therefore, each PE has capabilities for executing a different program independent of other PEs. Such machines can also be referred to as SPMD (Single Program Multiple Data) program models. These models are also called *asynchronous* programming model because it does not have a global clock to synchronize the various operations as in a SIMD machine.

MIMD model can be categorized as: *shared memory* and *distributed memory* computers. These models are discussed next.

3.1.4.1 Distributed Memory Machines

In distributed memory multiprocessors (Figure 3.3), all processors have their own local memory and operate independently. The communication between processors takes place via an interconnection network. The number of processors could vary from a few to several thousand in a distributed memory machine.

Parallel algorithm design techniques on such machines are slightly difficult compared to on shared memory machines. An algorithm designer is responsible for partitioning the data, distributing and mapping the data among the processors efficiently. Since there is no global memory on a distributed memory machine, all processors operate asynchronously and communication between processors is by means of message passing. Parallel Virtual Machine (PVM) and Message Passing Interface (MPI) are two message passing libraries. Intel Paragon [22], CM-5 [44], IBM SP/2, ncube [23] are representative examples of distributed memory machines.

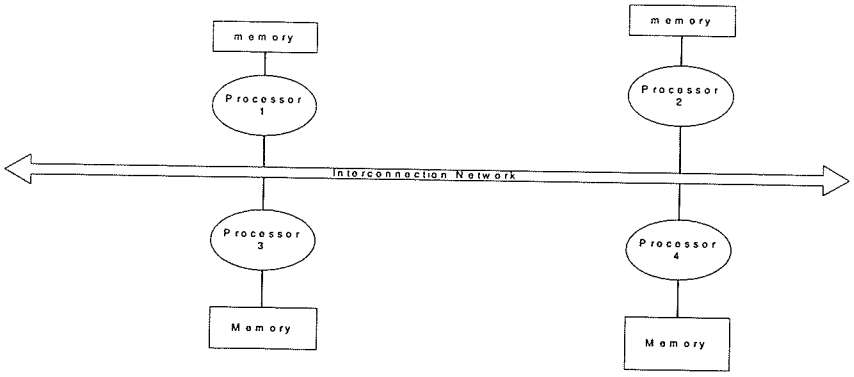

Figure 3.3 Distributed Memory Model

3.1.4.2 Shared Memory Machines

In shared memory machines (Figure 3.4), all the processors are connected to a single global memory; all the processor can access the global memory via an interconnection network [31]. Any changes to the data in the global memory by a processor are visible to all other processors.

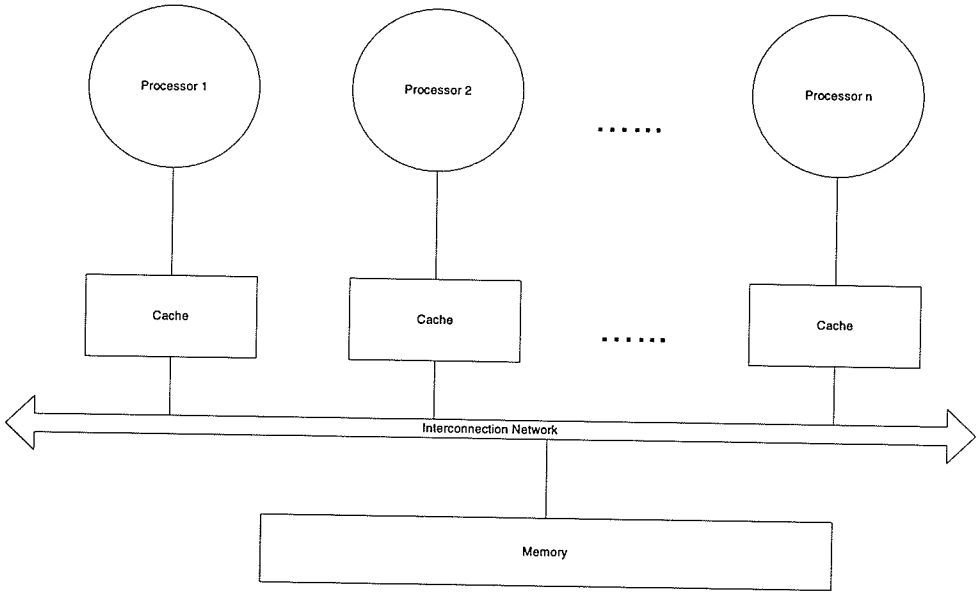

Figure 3.4. Shared Memory Architecture

To a user, it is easy to code parallel programs on such machines. For communication intensive problems, shared memory machines are preferable. Examples of representative machines are SGI Origin 3000 [52] and Sun's SMP's (Symmetric Multi-Processing. The drawback of shared memory machines is that synchronization primitives limit the speed of the computation since variables and resources in single address space may be shared. In general, to avoid this bottleneck, shared memory machines use high speed caches.

Shared memory MIMD models can be classified as: *Uniform Memory Access (UMA)*, *Non-Uniform Memory Access (NUMA)*, and *Cache Only Memory Architectures (COMA)* multiprocessors. The difference among these models is the way in which they share and distribute the memory resources.

The UMA model machines (Figure 3.5) are generally called as Symmetric Multiprocessors (SMP) machines. All processors have uniform access to memory. The entire memory is equally accessed by each processor. The interconnection network for UMA machines are a bus, crossbar switch or network. The number of processors in a SMP machine varies from 2 to 64.

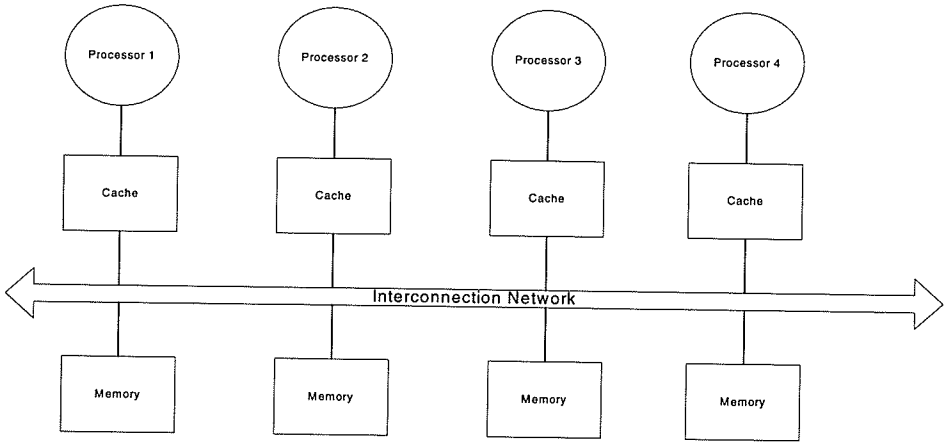

Figure 3.5. UMA model

The UMA machines, however, do not have good scalability. Caches are used to reduce access time to memory. However, this leads to cache coherency problems.

Two processors may have the same variable *x*. When processor 1 updates the value of *x* and writes the new value to memory, the other processor is unaware of this update and may keep the stale value. Therefore, protocols such as snooping protocol [24] are proposed to solve the cache coherency problem.

NUMA machines are designed to surpass the scalability limits of UMA machines. In NUMA machines (Figure 3.6), the shared memory is physically distributed amongst the processors called *local memory*. The local memories combined together appear as a shared global memory. Each local memory is closer to some processors than others. Each processor has access to all the local memories. Each local memory can be accessed faster by a processor than remote memories. NUMA machines are also called as *distributed shared memory* machines. The access to remote memories requires an interconnection network which connects processors and distributed memories. The interconnection network is a grid or hypercube. Therefore, NUMA machines scale to more processors.

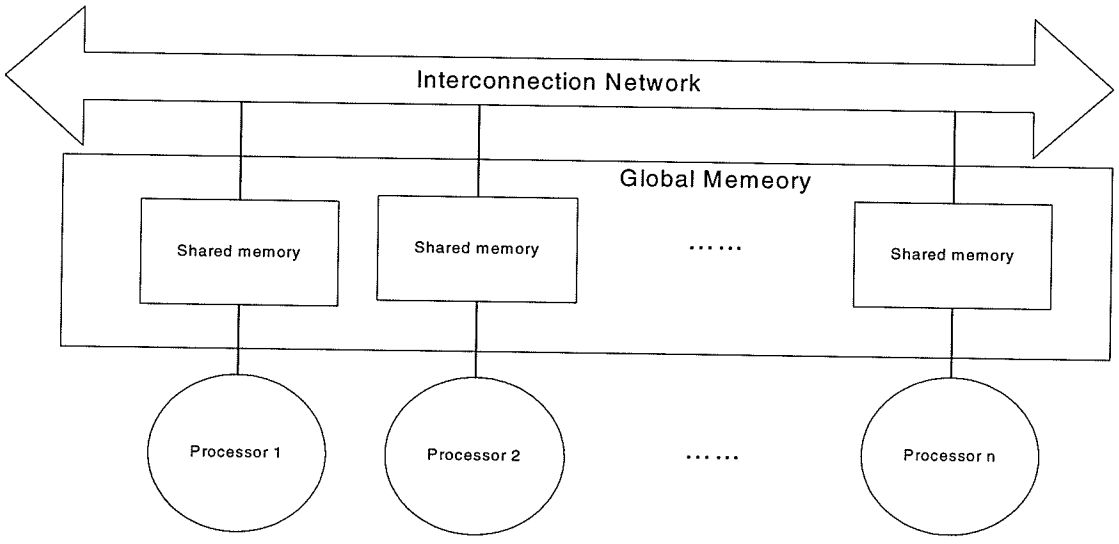

Figure 3.6. NUMA Model

In NUMA machines, cache coherency becomes important because the access time between local and remote memory grows with number of processors. However,

the sophisticated cache coherency protocols such as the directory based protocol are applied to solve the cache coherency problem. Therefore, NUMA machines actually mean Cache Coherent-NUMA (CC-NUMA) machines unless specially stated. SGI Origin 2000 and Sun Ultra HPC servers are some representative examples of the NUMA machines.

A COMA (Cache Only Memory Access) machine (Figure 3.7) is a special case of NUMA machines. The memories are converted to caches. All caches form a global address space. No main memory exists between the cache and disk layers. Processors access data in its own cache either from another caches or disk. Remote cache accesses are assisted by distributed cache directories. KSR-1 [54] is one example of COMA machines.

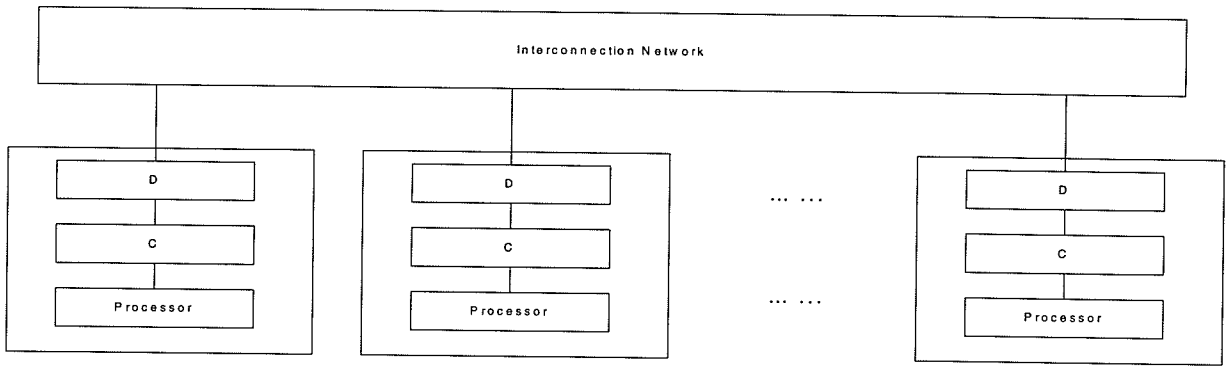

Figure 3.7. COMA Model

3.2 Multithreaded Paradigm

Massively parallel processors cause two fundamental latencies: *communication* latency and *synchronization* latency, as described by Arvind and Iannucci [14] in 1987. Communication causes a processor to idle until the necessary data is fetched from the destination processor. Synchronization between processors is required until the data is ready at the destination processor.

Many techniques can be used to reduce or tolerate latencies in hardware and software. The most promising technique is multithreading which tries to overlap computation with communication by means of threads (a thread is a set of instructions). The underlying multithreading principle is to keep processors always busy. In order to achieve this, the application must exhibit lots of parallelism. If parallelism exists, then it is up to the algorithm designer to efficiently determine the concurrent threads. Too many fine grained threads (few instructions per thread) may bombard the system. Coarse-grained threads may leave some processors idle or become more of SMD programming. The design and development of these threads depends on parallel algorithm designer. Multithreading has been enhanced in hardware, language or libraries. We briefly discuss them here.

3.2.1 Tera

Tera is designed as a scalable multithreaded architecture for supporting very fine grained multithreading [16] on a shared memory architecture. Each processor can execute 128 instruction streams (threads) simultaneously. Latency incurred in memory reference is hidden by fast context switching between the threads. All the processors have no data cache or local memory. They are connected via interconnection network to commodity memory and configured in a shared memory fashion. Tera supports a UNIX based operating system called TeraOS. The runtime system manages the interface between hardware, user programs and Tera`s operating system.

3.2.2 EARTH

The EARTH [33] is a multithreaded program dataflow model. A program running under EARTH is divided into threads that are scheduled atomically by using data

flow like synchronization operations. EARTH provides latency tolerance using efficient exploitation of fine grained parallelism by overlapping communication with computation. Each node in an EARTH system consists of an execution unit (EU), a synchronization unit (SU), which share the processors local memory. The *EU* executes fine grain threads, while the *SU* determines which thread is to be executed and handles communication between the processors. Programs running on EARTH are written in Threaded-C [62]. This is a extension of the ANSI-C language with multithreading instructions.

3.2.3 Cilk

Cilk is a fine grained multithreaded language [15] developed at MIT. Cilk is an algorithmic multithreaded language. Cilk is extremely simple. It consists of ANSI-C with three key constructs to indicate parallelism and synchronization. By removing the constructs, a Cilk program becomes a pure sequential code. Work and critical-path length are two parameters used to determine the performance of a multithreaded algorithm in Cilk. The scheduler is based on the concept of work stealing. This is very advantageous in building the Cilk code in a modular fashion as well as for performance evaluation.

3.2.4 PThreads

PThreads (POSIX threads) developed at Sun Microsystems is commercially available library. The threads library implementation follows a two-level multithreaded model. Programs are written using user level threads over possible lightweight processes. The kernel level threads are dispatched to the available processors by OS. This scheduling is transparent to the user. PThreads are widely used by parallel programmers, since it is an IEEE standard.

3.2.5 OpenMP

OpenMP stands for Open Specifications for Multi Processing. The OpenMP API is current industry standard for shared and distributed shared memory parallel programming directive [18]. This interface can be implemented in several programming languages like C, C++ and FORTRAN. Several compliers support OpenMP specifications. OpenMP avoids the overhead inherent in message passing systems.

In general, an OpenMP program executes as a Master/slave model. Master thread may fork many slave threads. The OpenMP program sequentially executes a single thread (called a master thread). When the master thread encounters the parallel constructs, a group of slave threads are created (fork operation) and each thread executes a copy of the body of sequential process enclosed within the parallel directive. The master thread will gain the control of the program when each slave thread completes computation (join operation).

In this thesis, we have used Java as the target language for all our implementation. In the next section, Java is explained.

3.3 Java

Java [48] is an object oriented language that has integrated multithreaded programming. Java allows concurrent programming for both single and multiprocessors system. Users can write platform independent multithreaded programs conveniently by means of threads. Java Virtual Machine (JVM) has a thread scheduler to monitor all running threads. When a Java class executes in a JVM, the main method of this class executes on a thread. Further threads can be constructed and started by a running thread. The code that executes on a separate thread is

isolated in the *run* method of a class that inherits from class *Thread* or implements interface *Runnable*. A thread is an object of an instance created from that kind class. A thread begins running when its *start* method is invoked. When the thread finishes its job, the *join* method can return back the control to the main program. The following is a example of multithread Java application.

```
public class MultithreadedJavaApplication {
      public static void main(String argv[]) {
        class sub_thread implements Runnable {
              public void run() {
                 // code to be executed on child thread
               }
        }
      sub_thread thread_1 = new sub_thread();
      sub_thread thread_2 = new sub_thread();
      Thread t_1 = new Thread(thread_1);
      Thread t_2 = new Thread(thread_2);
       t_1.start();
       t_2.start();
      // code to be executed on main thread after spawning child
      try{
       t_1.join();
       t_2.join();
      }
      catch(InterruptedException e) {
        //code to be executed when exception happens
       }
     // code to be executed on main thread after termination of child
     }
}
```

In many situations, separate, concurrently running threads share the same data. When the concurrently running code segments access the same object separately, the contention of the object occurs. The concurrently running codes are called *critical sections*. A critical section identified with the *synchronized* keyword can be a block or a method. The JVM associates a lock with every object. The Java run-time system automatically executes the acquisition and release of a lock with every object when entering and leaving critical sections. Therefore, the threads share

a common data sources, they must be synchronized in order to avoiding source contention.

3.4 Summary

In this chapter we presented a brief introduction to the Single Instruction Single Data (SISD), Multiple Instruction Single Data (MISD), Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD) parallel programming models. The language used for implementation is Java and is also described in this thesis. The reason behind using Java is that Java is considered the language of the internet. The thesis concentrates on internet applications and therefore Java is an ideal choice.

Chapter 4

Parallel Algorithms Design

This chapter describes the sequential algorithms of web log data preprocessing and association rule mining, followed by the parallel algorithm design presented in detail, with pseudo-codes and flowcharts.

4.1 Data Preprocessing

As mentioned in Chapter 2, data preprocessing is one of the most time consuming tasks in web mining. We first present the sequential data preprocessing algorithm, followed by its parallel version.

4.1.1 Sequential Data Cleaning Algorithm

In this section, we describe the improved data mining algorithm mentioned in Chapter 2.2.1. The sequential algorithm (called *WebLogCls*) to clean raw web log data is given below. In the data cleaning stage, irrelevant elements in raw web log data are removed. First, we build two list structures: *suffix list* and *robot list*. The suffix list consists of the suffix of image files. The robot list includes robots' IP addresses. Each entry of web log file is checked whether the suffix of its URL matches any values of the suffix list or its IP address matches any values in the robot list. The matched entries of web log file are removed from the results. The following is the detail of sequential *WebLogCls*:

WebLogCls
Input: *RL*=Raw_log_data
L =Parse(*RL*)
SL = {Suffix list};
RL= {Robot List}+"127.0.0.0";

```
1.  For (i=0;i<N;i++) /*N is the total number of web log records*/
2.    If (l[i] ∈ L).url ∈ SL Then
3.        insert i into D /*D is an array to store index of irrelevant web data */
4.      Else
5.          If (l [i]∈ L).ip∈ RL Then
6.              insert i into D
7.          End If
8.    End If
9.  End For
10. For (i=0; i<N; i++)
11.       If i ∉ D Then
12.          write(l[i].url, l[i].time, l[i].IP, l[i].agent) to T /* T is a file that stores
                                                          temporary cleaned web log data
*/
13.       Else next record
14.       End If
15. End For
```

The result of *WebLogCls* is a file *T*. The *T* is a set of entries. Each entry $t \in T$ consists of *url, time, IP*, and *agent* components. Other fields such as *rfc931, authuser, status, bytes,* and *refer* (mentioned in Chapter 2.2) in the web log data have been stripped out.

4.1.2 Parallel Processing of Data Cleaning

The *WebLogCls* algorithm is to eliminate irrelevant web log entities and identify web robots in the web log data. When using parallel computing techniques to process data cleaning, we horizontally divide the whole web log data into several small web log datasets of size *N/p* where *N* is the data size and *p* is the number of processors. Each processor executes the sequential *WebLogCls* algorithm and processes different web data sets. The following is the software model of parallel *WebLogCls* (Figure 4.1):

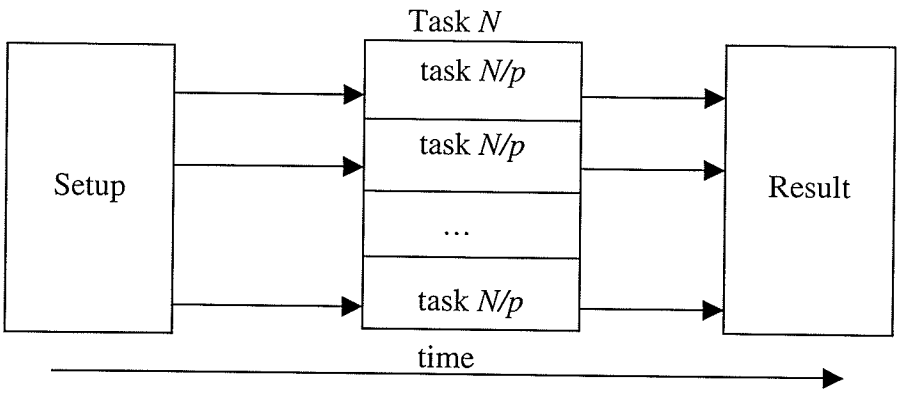

Figure 4.1. Parallel WebLogCls (p is the number of processors)

4.2 User Identification and Session Identification Algorithm

After data cleaning, user identification and session identification processes should be executed (as mentioned in Chapter 2.2). In this thesis, we combined user identification and session identification in order to reduce the time of scanning the whole dataset.

4.2.1 Sequential algorithm for User Identification and Session Identification

In this section, we describe the improved user identification and session identification algorithm mentioned in Chapter 2.2.3. The sequential algorithm (*WebLogIde*) identifies users and sessions of the web log data and is given below. Because there are no cookies in our web log data, we apply the heuristic proposed by Colley et al. [20] to identify users and use 30 minutes as time out to identify the sessions. The algorithm gets the first entry of web log data and puts this value into a list (called *User_list*) that stores the user's information (such as *IP, agent, timestamp*, and record *index)*. Then, the program sets the first value of *User_list* to a variable (*Current_user*) and uses *Current_user* to traverse the whole web log data and identifies the user and

session to match with the *Current_user* value. The sessions of *Current_user* is then identified by 30 minutes time out. If the program encounters a non-matched user (different IP and agent value), the program places the user's information into the *User_list*. This loop process will execute until the *User_list* is empty. The following is the sequential algorithm of user identification and session identification.

WebLogIde

```
Input : T = cleaned web log data
Variable: t[i] ∈ T
          User_list = Users and sessions are not identified
          Current_user = a record of User_list
          S= Session array
1. t[0] ∈T /* t[0] is the first entry in T */
2. put t[0].IP, t[0].agent, t[0].time and t[0].index into User_list
3. While Users_List is not empty
4.    Set Current_user = first item of User_list
5.    For (i = Current_user.index+1; i<N; i++) /*N is total number of entry in T
      */

6.       If t[i].IP=Current_user.IP and t[i].agent = Current_user.agent /* same
                                                                      user*/
7.          Then
8.             If t[i].time-Current_user.time <30 then /* period time is less
                                          30 minutes*/
9.                Put i into s[j]  /*s[j] ∈S */
10.            Else  /* period time is bigger than 30 minutes */
11.               create s[j++] /* create a new session*/
12.            End If
13.         Else  /*not same user */
14.            put t[i].IP, t[i].agent, t[i].time and t[i].index into User_list
15.   End If
16. End For
17. End while
```

4.2.2 Parallel Processing of WebLogIde

The sequential algorithm (*WebLogIde*) is developed for the user identification, session identification, and data format tasks. When identifying user sessions, we need to depend on the timestamp within each entry in the web log data. Therefore, we

cannot partition data into several small datasets. In this stage, we use multithreaded programming techniques to implement The sequential algorithm. In multithreaded algorithm, there are two steps.

In the first step, the main program uses the first entry of web log data to traverse the whole web log datasets. The first user and its related sessions can be identified using sequential *WebLogIde* algorithm. Meanwhile, non-matched users can be determined by the main program. Then, the main program stores the non-matched users' information (such as *IP, agent, timestamp*, and record *index*) into a list structure (*User_list*). Therefore, all different users' information is stored in the *User_list*.

Second step, the main program checks the *User_list* to determine it is not empty. If the *User_list* is not empty, the main program creates a set of user indicated number of threads. The main program assigns different values of *User_list* to each thread. Each thread uses its own *Current_user* value to execute the sequential flow of *WebLogIde* against different starting indices concurrently to determine user and session identification processes. Any non-matched entry of web log data encountered by a thread is ignored, and comparison moves to the next entry value of web log data. The result of parallel *WebLogIde* is a set of session values. These session values are stored in a global variable array (called *S*). Therefore, all threads output their results into the *S*. The threads synchronize to avoid race conditions and updating the same session value. At the completion of each thread task, the main program gets back the control of the program. The main program repeats the check of *User_list* until the *User_list* is empty. The Figure 4.2 shows the parallel *WebLogIde* algorithm model. In Figure 4.3, we use a flowchart to detail the multithreaded *WebLogIde* process.

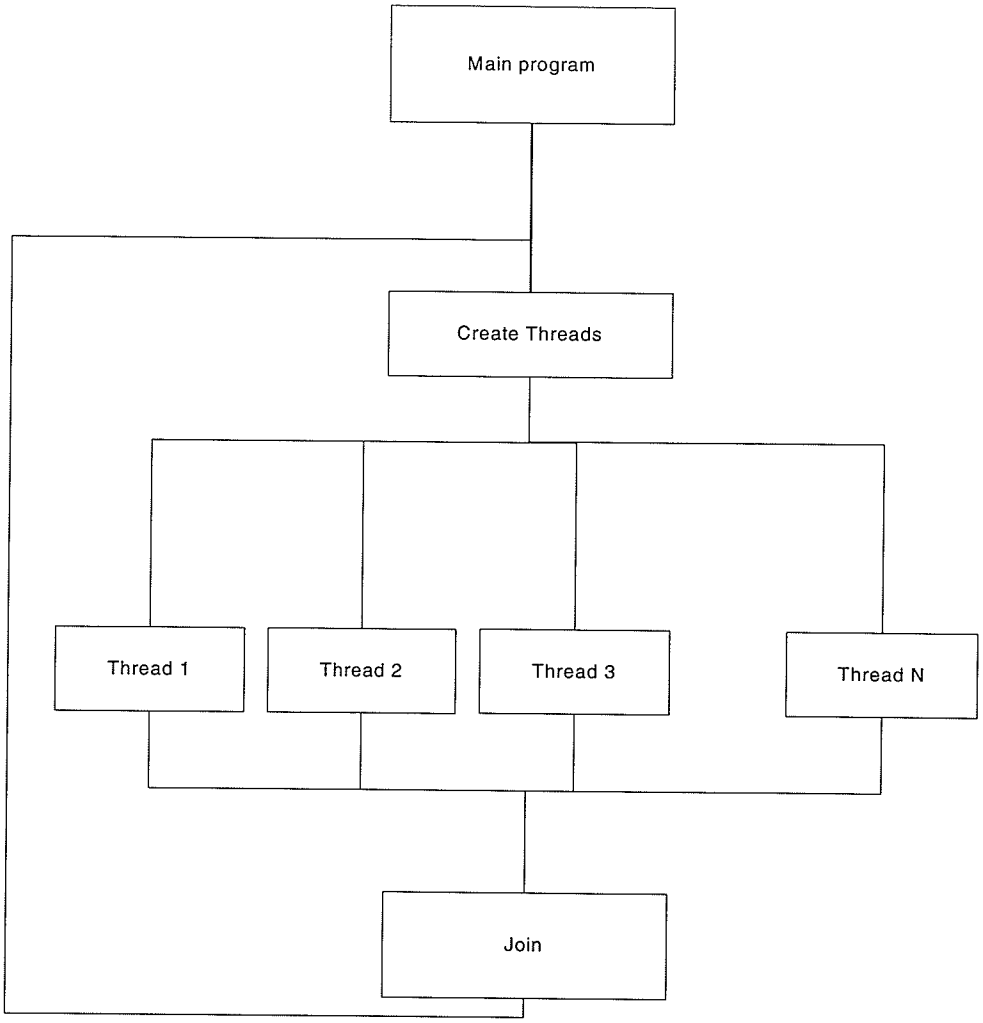

Figure 4.2. Parallel WebLogIde program model

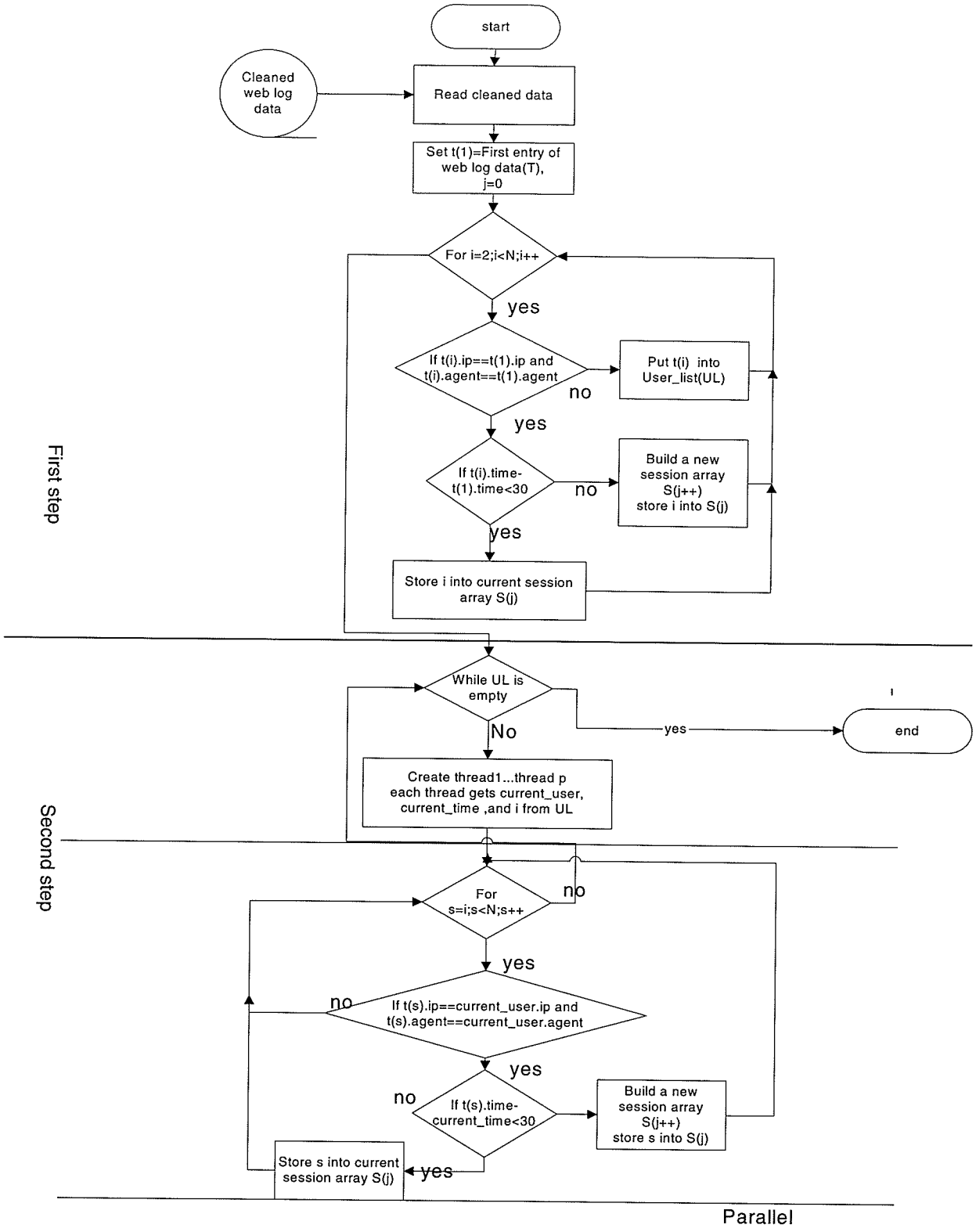

Figure 4.3 Parallel WebLogIde: Activity Flowchart

4.3 Association Rule Mining Algorithm

4.3.1 Apriori Algorithm

The Apriori [12] is a well-known association rule mining (ARM) algorithm. The Apriori algorithm serves as the foundation algorithm for most *ARM*, whether sequential or parallel algorithms. The Apriori applies an efficient method to generate candidate itemsets, thereby reducing the number of the candidate itemsets significantly.

In the first pass, frequent 1-itemsets (L_1) are generated by pruning those 1-itemsets whose support counts are below minimum support. For pass k>1, Apriori generates the candidate frequent k-itemsets (C_k) using the frequent (k-1)-itemsets (L_{k-1}); then it scans all transactions in the database to accumulate the support count of candidate k-itemsets. Those candidate itemsets whose support counts are not less minimum support are collected as frequent k-itemsets. The process is terminated until candidate itemsets are empty. The Apriori algorithm is described in [12] as shown:

Input: Database D

1) L_1=large 1-itemsets;
2) For(k=2;$L_{k-1} \neq 0$;k++) do begin
3) C_k=apripri-gen(L_{k-1}); /* Generate new candidates from L_{k-1}*/
4) For all transactions $T \in D$ do again
5) C_t=Subset(C_k,T); /* candidates contained in T*/
6) For all candidates $c \in C_t$ do
7) Count(c)=Count(c)+1;/* increase support count of C by 1 */
8) End For
9) L_k=$c \in C_k$ when Count(c)>= minimum support
10) End For
11) End For
12) Output $L_f = \cup_k C_k$

Generating candidate k-itemsets is based on the fact that any subset of (k-1)-itemsets (C_t) of a frequent k-itemset must also be frequent. In candidate generation

function, two different frequent (k-1)-itemsets which have same (k-2)-itemsets are jointed together to generate a candidate frequent itemset. This method prunes non frequent (k-1)-itemsets when generating candidate itemsets. By reducing the candidate itemsets that are known to be frequent, the number of passes required over the database is also reduced. Therefore, Apriori dramatically reduces computation, I/O, and memory requirement.

Detail of Apriori-gen function [12] is as follows:

Candidate Generation (apriori-gen function)

apriori-gen(L_{k-1});
To generate the set of all k-itemsets from L_{k-1}
join L_{k-1} and L_{k-1} such that,
$c_1 = (i_1, i_2, \ldots, i_{k-1})$ and $c_2 = (j_1, j_2, \ldots, j_{k-1})$ are joined if
$i_p = j_p$ for $1 <= p <= k-2$;
$c = (i_1, i_2, \ldots, i_{k-1}, j_{k-1})$; /* new candidate c*/
Add c into candidate itemsets C_k.

Apriori uses a hash tree data structure to store the candidates [12]. The hash tree structure provides an efficient method to scan the transactions. In a hash tree, itemsets are stored in the leaves; the interior contains a hash table. The algorithm does not store all transactions in the main memory, only the hash tree stays in the memory.

Figure 4.4 shows one candidate hash tree with candidates of length 2. When a candidate itemset is generated, the items in this candidate are stored in a sorted order. When a candidate itemset is inserted in the hash tree, the itemset goes down from the root by hashing each item at interior nodes until reaching a leaf. The candidate hash tree function is shown as follows:

CreateCandidateTree:
Input: *CI* =candidate itemsets

1. *L*=len(*CI*) /*get the number of itmesets in *CI*/
2. *Root*= new Hash_node()
3. For (*i*=1; *i*<*L*; *i*++)
4. *canda*=*CI*(*i*) /* get i*th* itemset of *CI* */
5. Call GeneHashtree (1, *Root*, *canda*)
6. End For

```
GeneHashtree (int i, hash_node h, string s) /*generate Hash tree function*/
  Input variable: i = the depth of hash tree
          h = hash_node /*a data structure*/
          { status , /* status=L means a Leaf, status=N means hash node*/
           HT, /* hash table*/
           itemset_list /* a Vector*/
          }
          s = itemset
 1. N= len(s)     /* get length of s */
 2. If (i= =N)     /* reach the a leaf */
 3.    set h.status=L  /* set h as a leaf*/
 4.    If h.itemset_list is null
 5.       Then create a itemset_list Vector for h
 6.            add itemset s into  h.itemset_list
 7. Else   /* not reach the leaf */
 8.    If  h.HT is a Null table
 9.       create a new HT for h
10.    End If
11.    If  (h.HT contains key value of ith item of s)
12.       Then
13.          set h.status=N  /* set h as a hah node*/
14.          set  h = hash_node having keyword ith item of s in h.HT
15.          Call GeneHashtree (i+1,h,s) /* recursive call GeneHashtree */
16.       Else
17.          create a new hash_node( hn)
18.          put hn  into h.HT based on ith item of s
19.          Call GeneHashtree(i+1, hn, s) /*recursive call GeneHashtree*/
20.    End If
21. End If
```

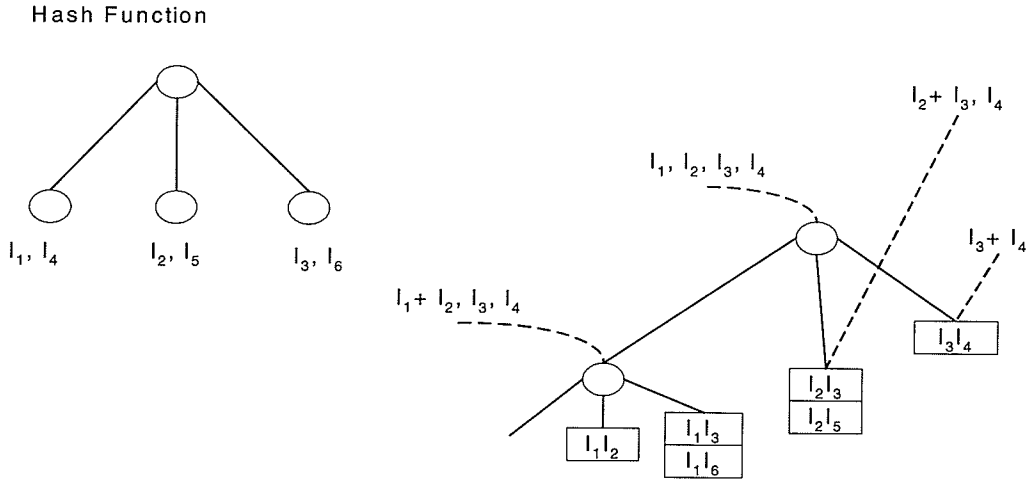

Figure 4.4. Subset Function Traversing the Hash Tree

A subset function is used to traverse the hash tree from the root in order to find all candidates in a transaction t. This process executes recursively until a leaf is reached. All the candidates are checked and the support count updated accordingly. In Figure 4.4, subset function traverses the hash tree from the root using a transaction $\{I_1, I_2, I_3, I_4\}$. The item I_1 is hashed to the left child node and following $\{I_2, I_3, I_4\}$ is recursively hashed to the left child node. The item I_2 is hashed to middle child node and two candidate itemsets ($\{I_2, I_3\}$, $\{I_2, I_5\}$) are checked whether they are contained in the whole transaction $\{I_1, I_2, I_3, I_4\}$. The item I_3 is hashed to the right child node of the root and the transaction is checked whether it contains the candidate itemset $\{I_3, I_4\}$. The Figure 4.5 shows the subset function traverses the left child node of the root in the $\{I_2, I_3, I_4\}$. The item I_2 is hashed to the middle child node and the transaction is checked against the candidate itemset $\{I_1, I_2\}$. The item I_3 is hashed to the right child node and the two candidate itemsets (e.g., $\{I_1, I_3\}$ and $\{I_1, I_4\}$) are checked in the transaction. The pseudo-code of subset function is shown as follows:

```
Input: t= transaction
       s = the position of item in the transaction t
       HTRD = hash_node /*a data structure*/
            { status , /* status=L means a Leaf, status=N means hash node*/
             HT, /* hash table*/
              itemset_list /* a Vector*/
            }
Traversetree(s, HTRD, t)

1.  If  HTRD.status=L /* a leaf */
2.     L= size (HTRD.itemset_list) /*L is the number of itemsets in this hash node*/
3.        For (i=0; i<L;i++)
4.          itemset= ith itmeset in HTRD.itemset_list /*get an itemset
                                                        from this hash node*/
5.           If  t contains the last item of itemset /*find out a matched itemset
                                                      in transaction t*/
6.           Then
7.             Count(itemset)= Count(itemset)+1 /*itemset support count add 1*/
8.           End if
9.        End for
10.  Else  /* hash table*/
```

```
11.    for( j=s+1; j<len(t); j++) /*len(t) is the length of the transaction t */
12.      If HTRD.HT contains the jth value of t  /*find out a matched item in t */
13.        Then
14.          CALL Traversetree(j, HTRD, t) /*recursive call
                                              Traversetree(s, HTRD, t)  */
15.      End if
16.    End for
17.  End if
```

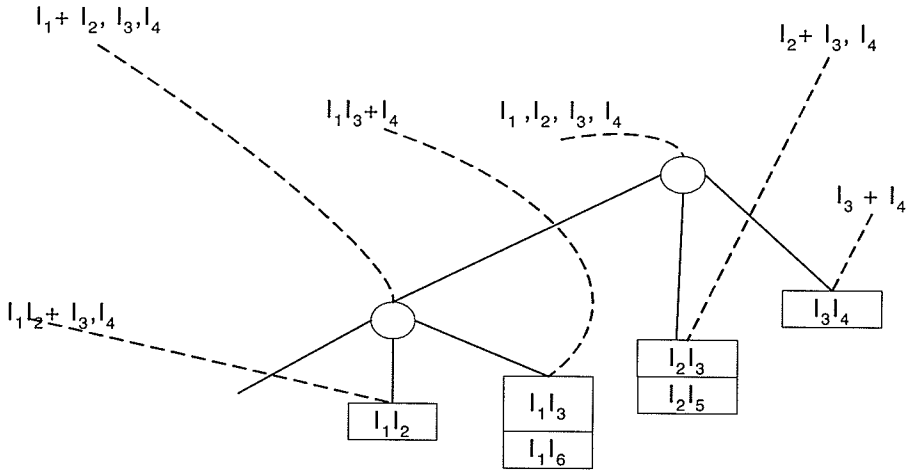

Figure 4.5 Subset Function Operating in the Left Child Node

4.3.2 Parallel Association Rule Mining

The current parallel and distributed ARM algorithms are based on Apriori. In general, the parallelism of ARM can be categorized as: data parallelism and task parallelism [64]. The difference of the two paradigms of parallelism is whether the candidate itemsets are distributed across the processors or not. In the data parallelism paradigm, each processor calculates the support counts of entire candidates. In the task parallelism paradigm, the candidate itemsets are partitioned across the processors, each processor counts the support counts of candidates distributed.

In this thesis, the Count Distribution (CD) [11] data parallelism of ARM is implemented. Note that this algorithm was originally proposed for the distributed memory parallel machine. However, we developed the parallel algorithm and

implemented on a shared memory machine. In the CD parallelism paradigm model, the dataset is partitioned among the processors and the candidates are duplicated on all processors. Each processor first scans its own partitioned transactions of the database to compute the local support counts of all candidates. After computing local support counts of all candidates, all processors calculate the global support count of all candidates in the database by exchanging the local support counts (Global Reduction). Following that, frequent itemsets are computed by each processor independently. CD repeats these steps until no new candidate is generated. The CD paradigm is illustrated in Figure 4.6. The four transactions (N) are partitioned across the three processors with each processor having N/p transactions. The three candidate itemsets in the second scan are duplicated on each processor. The local support counts are shown after scanning the local databases.

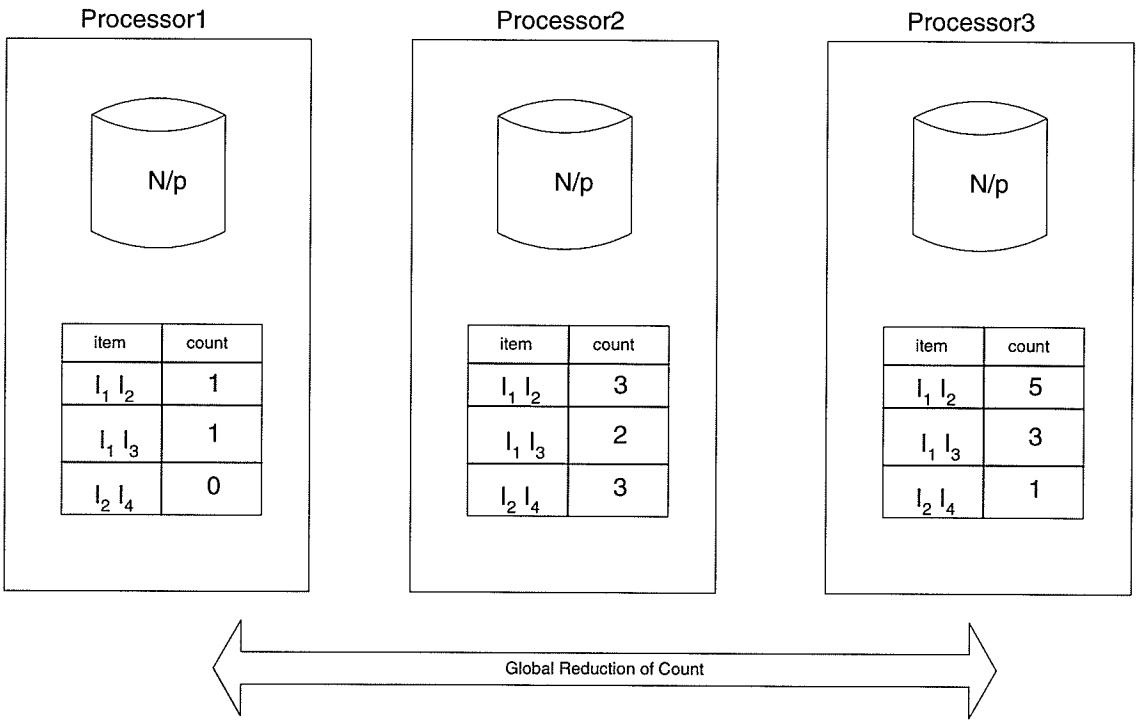

Figure 4.6 Data Parallelism Paradigm

We follow the CD paradigm to implement parallel ARM algorithm under a shared memory system. We logically split the whole transactions of URLs into several partitions ($T_1...T_p$) among processors, where, $T_1...T_p = N/p$, N is the entire number of transactions and p is the number of processors. The master program generates the candidate (called common candidates) C_k based on frequent itemsets L_{k-1} that is created at the end of k-1 iteration. The common candidate hash tree (HT) is created by the master program. Each processor traverses its local transactions to compute the support count for each common candidate. Due to the race condition encountered when different processors update counts of the same candidate, in this thesis, we developed a global candidate count structure. We use the synchronized mechanism of Java to prohibit more than one processor to increase the same count field simultaneously. After all processors finish traversing their own partitioned data, the master program gets back the control of the program. The frequent itemsets are generated by the master program following the global candidate counts. These steps are repeated by the program until no new candidates are generated. We present the flowchart of the parallel ARM algorithm in Figure 4.7.

4.4 Summary

In this chapter, the sequential and parallel algorithm for data preprocessing, user/session identification and association rule mining will be implemented on a shared memory using Java in the next chapter

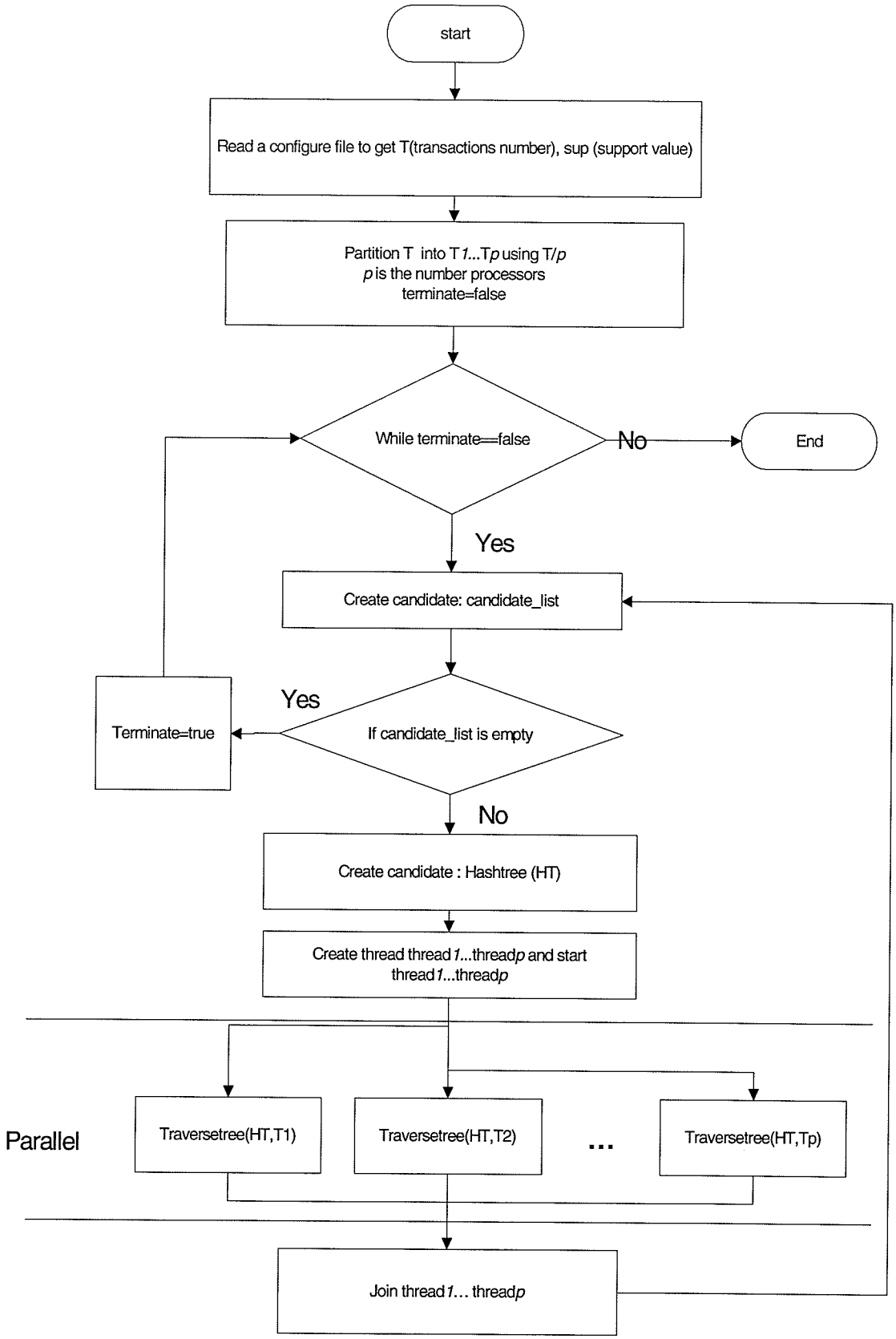

Figure 4.7 Parallel Programming ARM Flowchart

Chapter 5

Experimental Results

This chapter presents the results of web usage mining, which includes web log data preprocessing and user pattern discovery. The algorithm performance is evaluated through speedup, calculated as

$$Speedup = \frac{Execution\,time\,on\,one\,\,processor}{Execution\,time\,on\,\,p\,\,processor}$$

5.1 Experimental Results of Web Log Data Preprocessing

In this section, we provide results for web log data cleaning and web log user identification processes.

5.1.1 Web Log Data Cleaning Result

To determine the performance of parallel data mining, four different bytes of records from 20.4M raw data [1] was extracted and cleaned by various numbers of processors. The outcome of the records obtained after data cleaning for each set of data is shown in Figure 5.1. For example, the entire web log data includes 210496 records. After data cleaning process, the cleaned web log data includes 160905 records Figure 5.1 shows the execution times of data cleaning on various number of web log data sizes. As expected, the execution time gradually decreases as the number of processors increases. The speedup on 8 processors to obtain a cleaned data records for 200,000 is around 2.5. The reason behind this low speedup is in I/O contention. That is, at the end of data cleaning each thread writes the output result to a

file. Though each thread works on its sub data, all threads write to the same file. Therefore, there is a some synchronization barrier at this point.

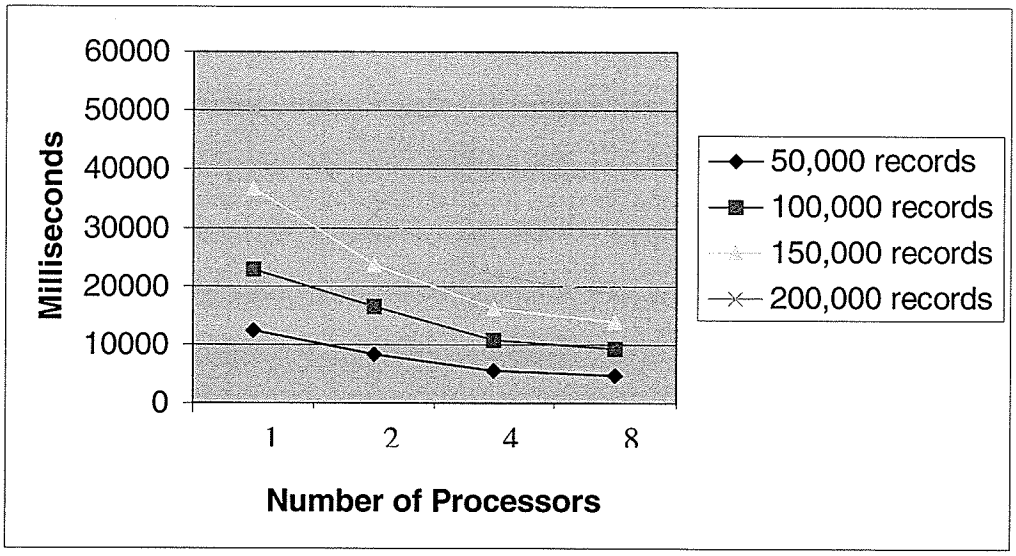

Figure 5.1. Performance of the Parallel Data Cleaning

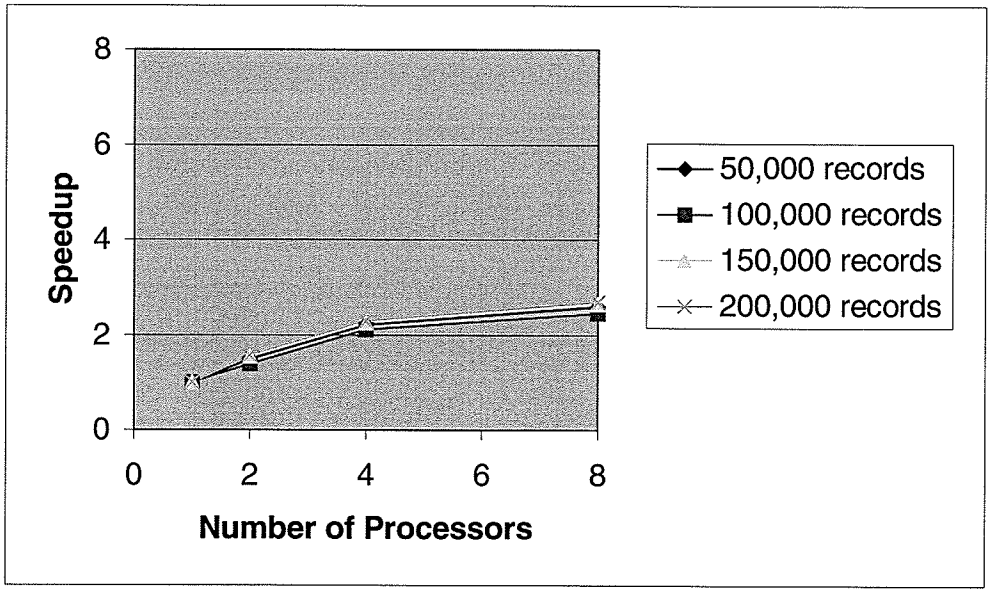

Figure 5.2. Speedup of Parallel Data Cleaning

5.1.2 User Identification Results

In this section, we experiment with the parallel algorithm *WebLogIde*, discussed in chapter 4. This algorithm uses the data obtained after data cleaning as input. The output data of this algorithm is the formatted web log data with user identification and session numbers. Figure 5.3 shows the performance of the algorithm in various number of processors. For 160905 records we see that there is a sharp decrease in execution time from one to two processors and to 8 processors. The speedup, however, is lower than two on 8 processors. The reason is that the threads share the data stored in shared memory. For example, we created 8 threads, and each thread accesses the same data in shared memory for different user identifications. We do not partition data because a user *x* record could be located in any location of the shared memory.

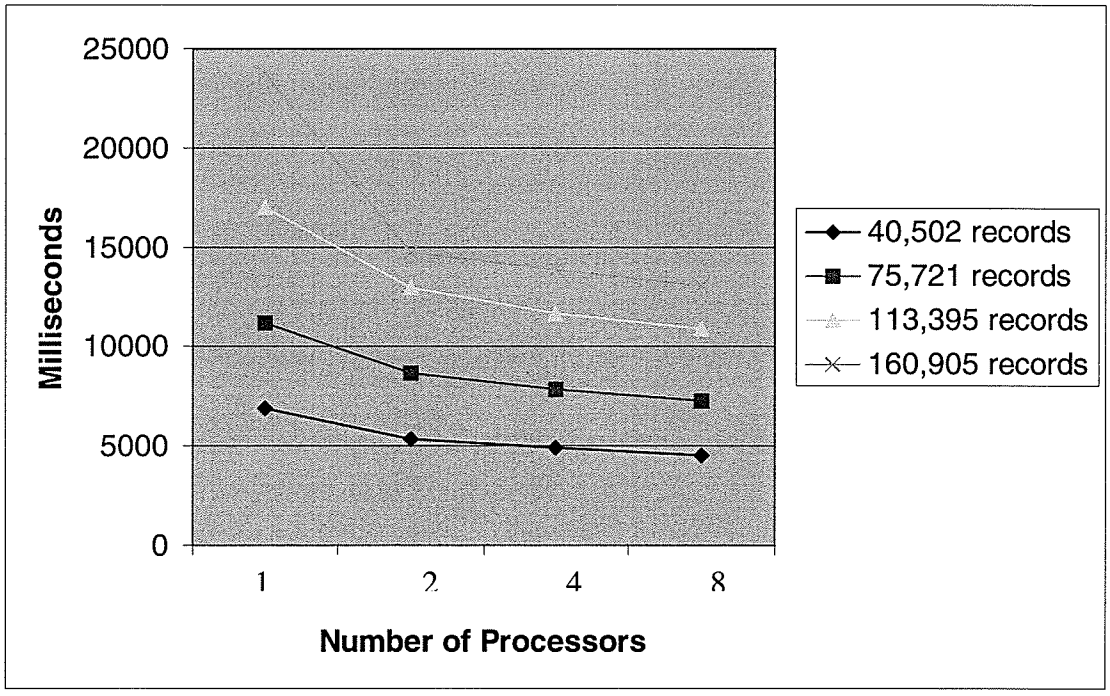

Figure 5.3. Performance of the Multithreaded WebLogIde

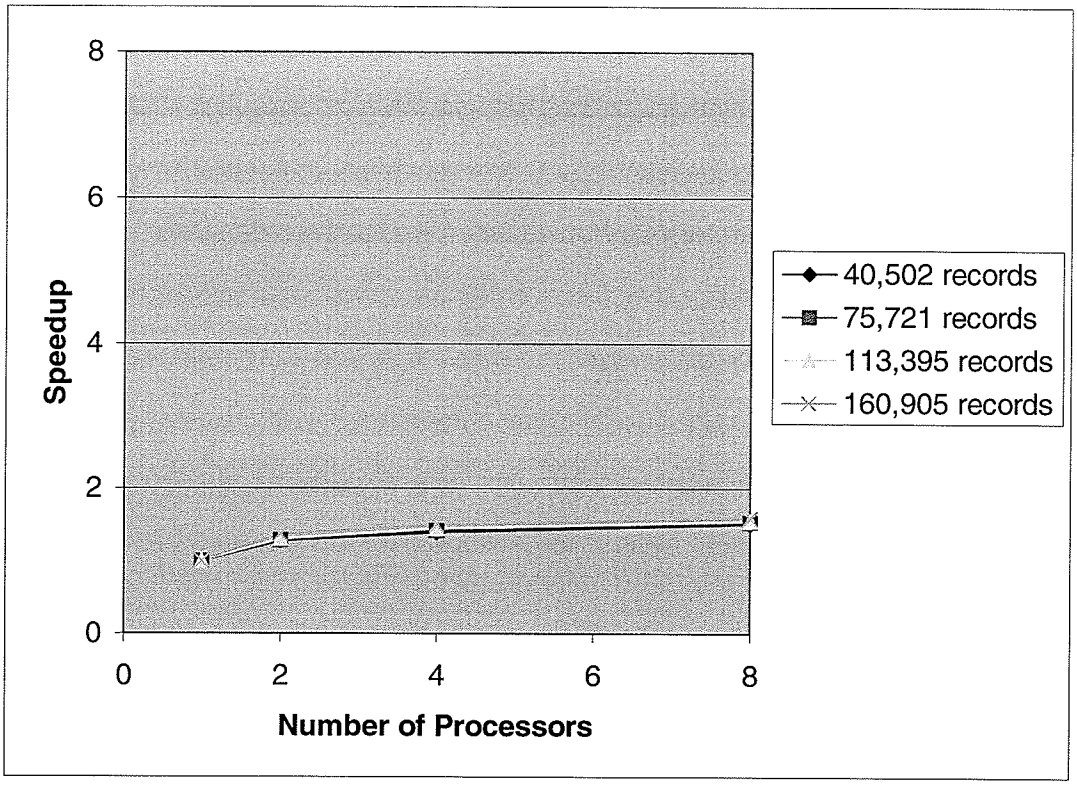

Figure 5.4 Speedup of Multithreaded WebLogIde

5.2 Association Rule Mining Results

The performance results of the parallel ARM algorithm is shown in Figure 5.5 and Figure 5.6. As the number of processors increase, there is a sharp decrease in execution time for various records. The speedup on 8 processors is about 3 for 16742 records. This result is encouraging for large data sizes.

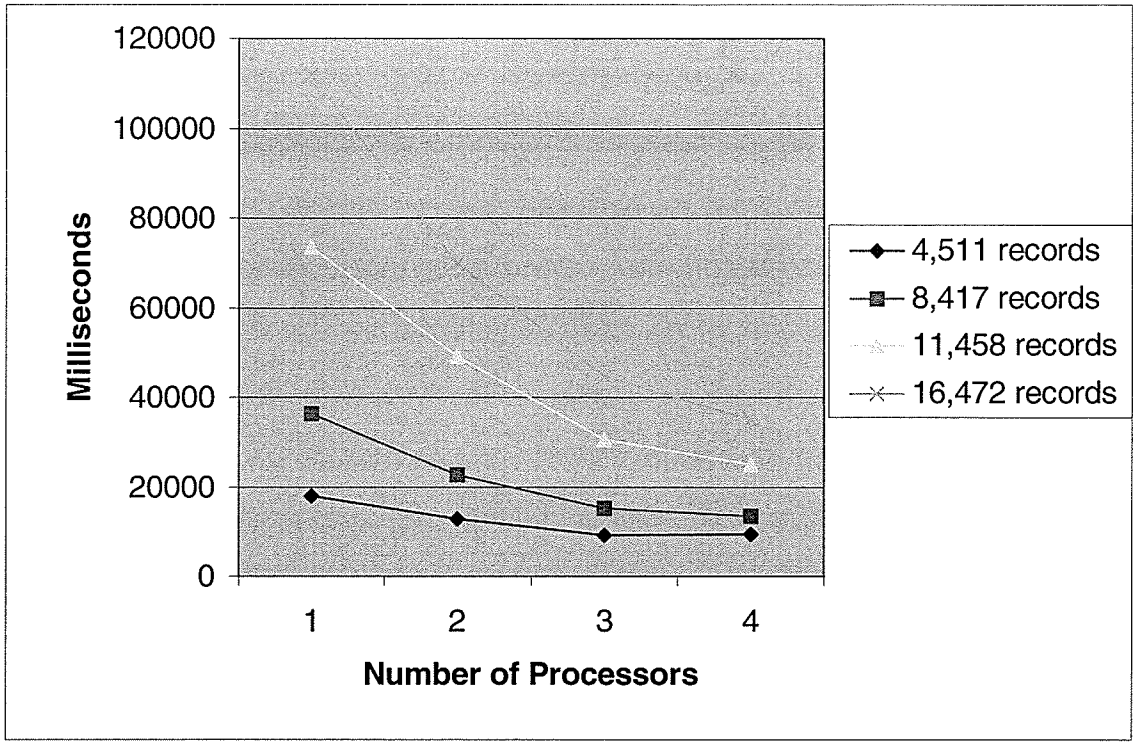

Figure 5.5. Performance of Parallel ARM

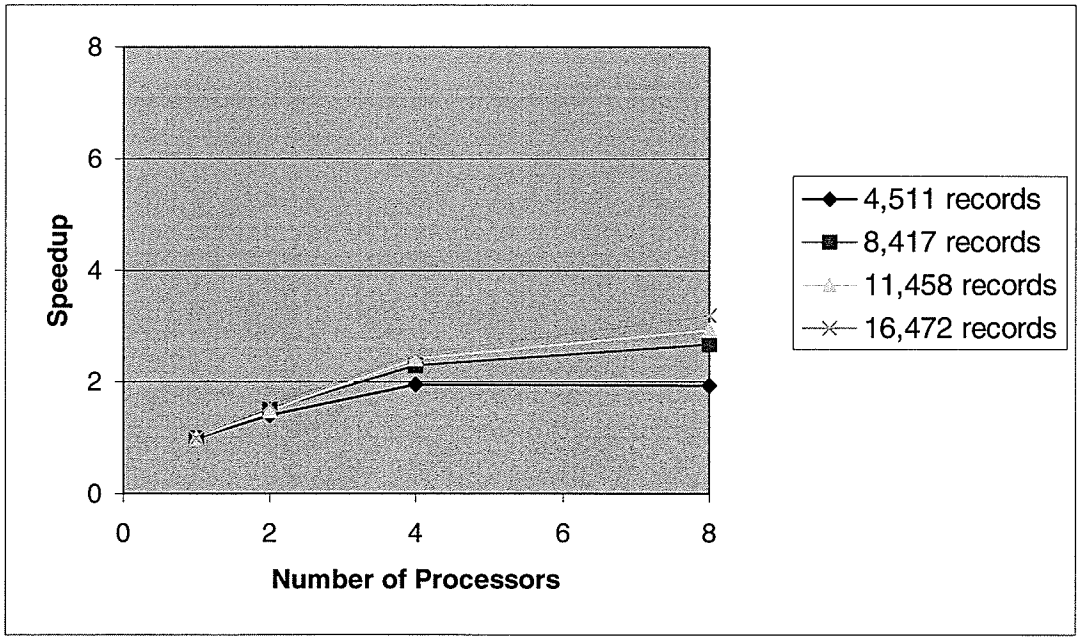

Figure 5.6. Speedup of Parallel ARM

5.3 Overall Web Usage Mining Results

Figure 5.7 and Figure 5.8 show the entire performance results of web usage mining (data cleaning, user identification, ARM). Figure 5.7 indicates that as the number of processors increases, the execution time decreases. This reduction can be seen for 200,000 records where the execution time for 4 processors is approximately 7 seconds compared to 190 seconds for one processor. The overall speedup is about two for 4 processors and only close to 3 for 8 processors. This is due in part to the contention in accessing data and synchronization between the various stages of web usage mining.

Finally, Figure 5.9 shows the performance results with respect to various data sizes. This figure indicates that the web usage mining algorithms is very scalable.

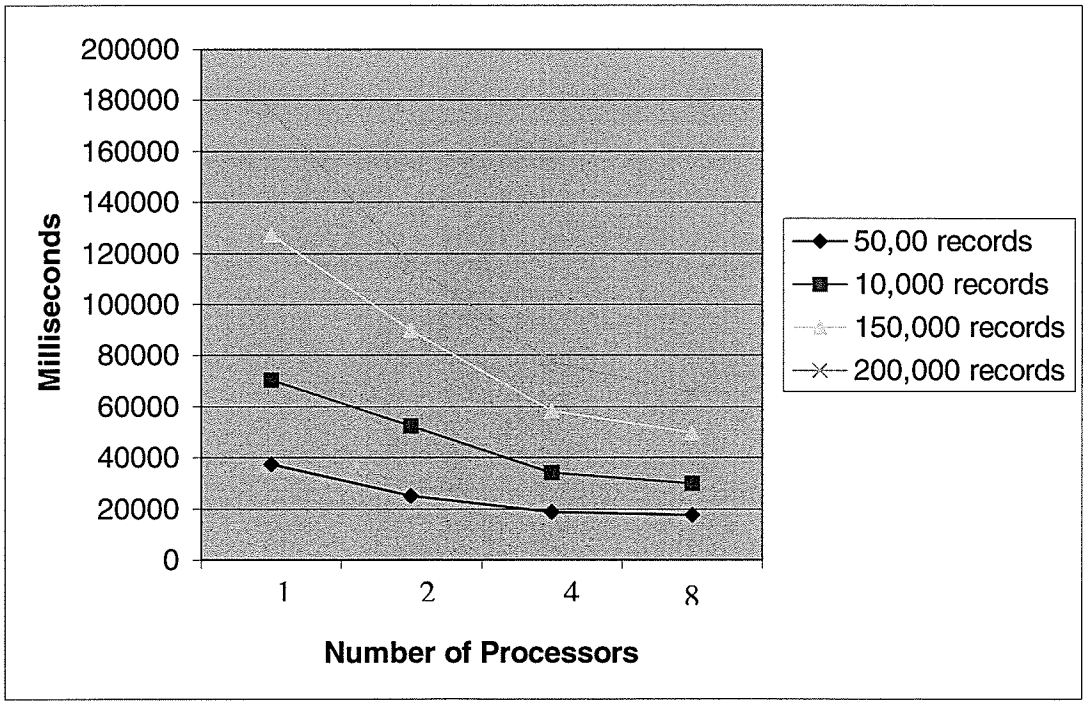

Figure 5.7. Performance of web usage mining

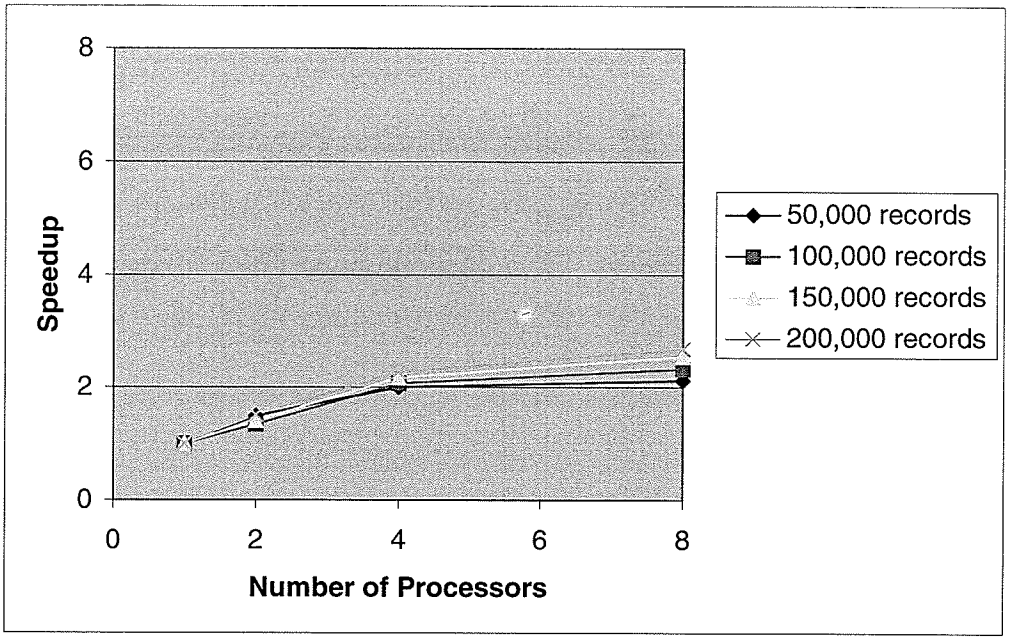

Figure 5.8. Speed up of web usage mining project

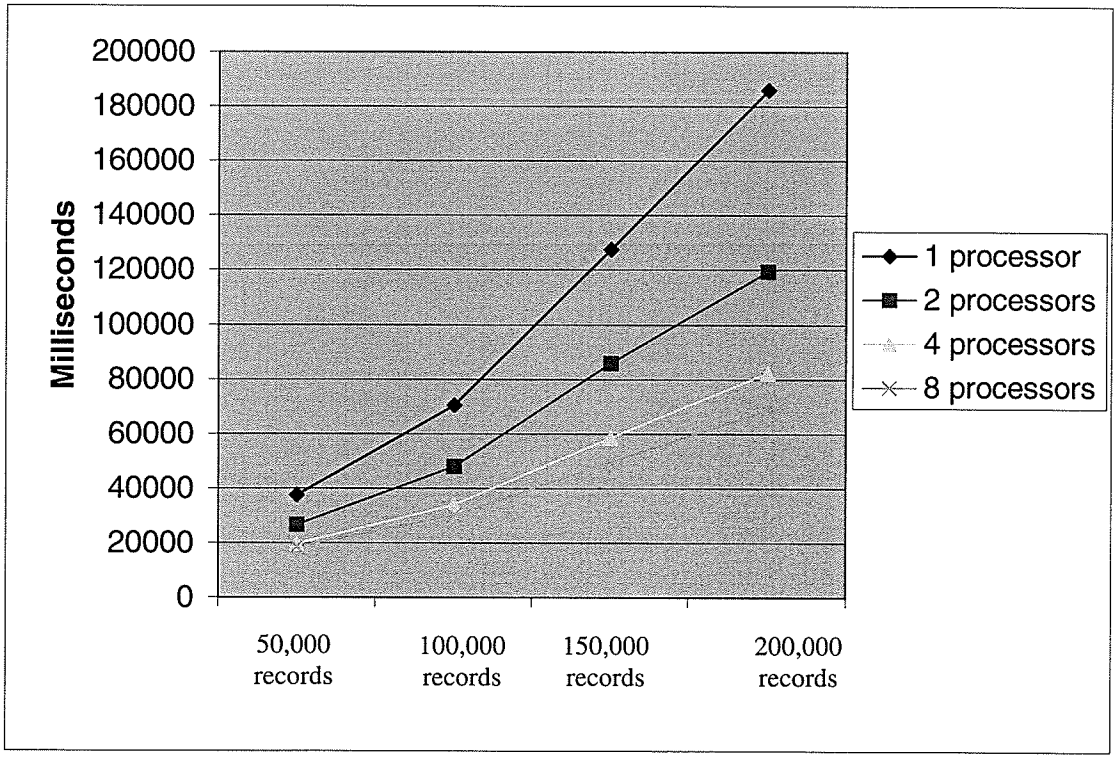

Figure 5.9. Performance at varying data sizes.

5.4 Summary

In this chapter, we discussed the experimental results of the web usage mining parallel algorithm. The overall speedup is about two on four processors and close to three on 8 processors. For large data sizes, we do see significant reduction in execution time as we increase the number of processors. This result is encouraging and motivates future research in parallelization of web usage mining.

Chapter 6

Summary and Conclusions

In this thesis, parallel computing techniques were used to improve the performance of web usage mining. In the data preprocessing or cleaning stage, we horizontally divide the whole web log data into several small web log datasets of size of N/p where N is the data size and p is the number of processors. The sequential web log data cleaning algorithm is executed by each processor. The next stage is user identification and session identification process. In this stage, we use multithreaded program mining techniques to implement the use/session identification algorithm. The output of the algorithm is a set of session values. The parallel association rule mining (ARM) algorithm is based on Apriori. In this thesis, we implemented the count distribution data parallelism of ARM.

The parallel algorithms are implemented using Java on 8 processors. The raw web log data [1] includes 210496 records. The speedup after web log data cleaning is 2.5 to obtain a cleaned data record from 200,000 records. There is some synchronization that could not be eliminated that creates a bottleneck. The experimental results of user/session identification with 8 threads give a speedup of lower than two. This is a race condition among the threads. For large data size, the parallel ARM algorithm produces a speedup of 3.

In summary, the results of web usage mining (data cleaning, user/session identification, and ARM) indicates that as the number of processors increases, the execution time decreases. This reduction can be seen for 200,000 records where the execution time for 4 processors is 7 seconds compared to 190 seconds for one

processor. The overall speedup is about 2 for 4 processors and only close to 3 for 8 processors. This is due in part of the contention in accessing data and synchronization between the various stages of web usage mining.

6.1 Future Work

As future work, we propose the following:

1) The web robot's identification affects the accuracy of web usage mining's results. In this thesis, we used the simplest method of identifying and removing robots session based on bots' user agent or IP (two fields) in web log data. The robots' IP addresses are dynamic. Therefore, using this simplest method can not remove all web robots' records from the web log data. Finding out dynamic web robots is a further interesting research topic.
2) The result of web usage mining in this thesis is a set of frequent itemsets. In order to provide most information to web users, integrating web contents with web pages is a necessary method. Most web pages are dynamic and contain many items. Combining web contents with web pages, we can extract more exact information about users' behaviours. Therefore, integrating web usage mining and content mining can achieve further gains for web personalization.

References

[1] http://sandalwood.cs.umanitoba.ca.

[2] Access Log Analyzer. http://www.uu.se/Software/Analyzers/Access-analyzers.html.

[3] Altavista Search Engine. http://www.altavista.com.

[4] Common Log Format. http://www.w3.org/Daemon/User/Config/Logging.html

[5] Google Search Engine. http://www.google.com.

[6] HTTP - Hypertext Transfer Protocol. http://www.w3.org/Protocols/

[7] Keynote Internet Performance Authority. http://www.keynote.com/.

[8] mySimon. http://www.mysimon.com

[9] WebLogs. http://www.cape.com.

[10] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data,* Washington, D.C., United States, pages 207-216, May 1993.

[11] Rakesh Agrawal, John C. Shafer. Parallel Mining of Association Rules. *IEEE Transaction On Knowledge and Data Engineering,* 8(6): 962-969, December 1996.

[12] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Database. In *Proceedings of 20th International Conference on Very Large Data Bases, (VLDB),* Santiago, Chile, pages 487-499, September 1994.

[13] Suhail Ansari, Ron Kohavi, Llew Mason and Zijian Zheng. Integrating E-Commerce and Data Mining: Architecture and Challenges. In *ICDM'01: The 2001 IEEE International Conference on Data Mining,* San Jose, California, USA, pages 27- 34, November 2001.

[14] Arvind and Robert A. Iannucci. Two fundamental issues in multiprocessing. In *Proceedings of 4th International DFVLR Seminar on Foundations of Engineering Sciences on Parallel Computing in Science and Engineering,* Bonn, Germany, pages 61-88, June 1988. Springer-Verlag New York, Inc.

[15] Robert D. Blumofe, Christopher F. Joerg, Bradley Kuszmaul, Charles E. Leiserson, Keith H. Randall and Yuli Zhou. Cilk: An Efficient Multithreaded Runtime System, In *Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Santa Barbara, California, pages: 207-216, July 1995.

[16] Sharon Brunett, Jhon Thornley and Marrq Ellenbecker. An Initial Evaluation of the Tera Multithreaded Architecture and Programming System Using the C3I Parallel Benchmark Suite, *In Proceedings of the Supercomputing '98*, Orlando, Florida, pages 1-19, November 1998.

[17] Alex G. Buchner, M. Baumgarten, Sarabjot S. Anand, Maurice D. Mulvenna and J.C. Hughes. Navigation Pattern Discovery from Internet Data. In *Proceedings of the Web Usage Analysis and User Profiling Workshop,* San Diego, California, August 1999.

[18] Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald and Ramesh Menon. *Parallel Programming in OpenMP.* Morgan Kaufmann, San Francisco, CA. October 2001.

[19] Edith Cohen, Balachander Krishnamurthy and Jennifer Rexford. Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters. In *Proceedings of ACM SIGCOMM,* Vancouver, Canada, pages 241-253, September 1998.

[20] Robert Cooley, Bamshad Mobasher and Jaideep Srivastava. Data Preparation for Mining World Wide Web Browsing Patterns. *Knowledge and Information Systems,* 1(1):5-32, February 1999.

[21] Robert Cooley, Jaideep Srivastava and Bamshad Mobasher. Web Mining: Information and Pattern Discovery on the World Wide Web. In *9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97),* New port Beach, California, pages 558-567, November 1997.

[22] Intel Corporation. PARAGON OSF/I, Users guide. April 1993.

[23] NCUBE Corporation. NCUBE Users Handbook. Beaverton, Oregon, USA, October 1987.

[24] David E. Culler, Jaswinder Pal Singh and Anoop Gupta. *Parallel Computer Architecture : A Hardware/Software Approach.* Morgan Kaufmann Publishers, San Francisco, August 1998.

[25] Ayhan Demiriz and Mohammed J. Zaki. webSPADE: A Parallel Sequence Mining Algorithm to Analyze the Web Log Data. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02),* Maebashi City, Japan, pages 755-758, December, 2002.

[26] Magdalini Eirinaki and Michalis Vazirgiannis. Web Mining for Web Personalization. *ACM Transactions on Internet Technology.* 3(1):1-27, February 2003.

[27] Oren Etzioni. The World-Wide Web: Quagmire or Gold Mine? *Communications of the ACM,* 39(11):65-58, November 1996.

[28] Usama M. Fayyad, Gregory Piatetsky-Shapiro and Padhraic Smyth. From Data Mining to Knowledge Discovery: an overview. *Advances in Knowledge Discovery and Data Mining,* pages 1-34. American Association for Artificial Intelligence, March 1996.

[29] William J. Frawley, Gregory Piatetsky-Shapiro and Christopher J. Matheus. Knowledge Discovery in Databases - An overview. *AI Magazine,* 13:57-70, Fall 1992.

[30] Xiaobin Fu, Jay Budzik and Kristian J. Hammond. Mining navigation history for recommendation. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces,* New Orleans, LA, pages 106-112, January 2000.

[31] Ananth Grama, Anshul Gupta, George Karypis and Vipin Kumar. *Introduction to Parallel Computing.* Pearson Education Limited, Harlow, England. February 2003.

[32] Dan R. Greening. Building Consumer Trust With Accurate Product Recommendations. LikeMinds White Paper LMWSWP-210-6966, 1997.

[33] Laurie J. Hendren, Xinan Tang, Yingchun Zhu, Shereen Ghobrial, Guang R. Gao, Xun Xue, Haiying Cai and Pierre Ouellet. Compiling C for the EARTH Multithreaded Architecture. *International Journal of Parallel Programming.* 25(4):305-338, August 1997.

[34] V. Jacobson, C. Leres, and S. Mclanne. Tcpdump, available via anonymous ftp to ftp.ee.lbl.gov, June 1989.

[35] Heaton Jeff. *Programming Spiders, Bots, and Aggregators in Java.* Sybex Book, February 2002.

[36] Anupam Joshi and Raghu Krishnapuram. On Mining Web Access Logs. In *ACM Special Interest Group On Management Of Data (SIGMOD) Workshop on Research Issues in Data Mining and Knowledge Discovery,* Dallas, Texas, pages 63-69, May 2000.

[37] Masaru Kitsuregawa, Takahiko Shintani and Iko Pramudiono. Web Mining and Its SQL Based Parallel Execution. In *Proceedings of the Workshop on Information Technology for Virtual Enterprises,* Queensland, Australia, pages 128-134, January 2001. IEEE Computer Society Press.

[38] Ron Kohavi, Llew Mason, Rajesh Parekh, Zijian Zheng. Lessons and Challenges from Mining Retail E-Commerce Data. *Machine Learning,* 57(1-2): 83-113, October 2004.

[39] Ron Kohavi and Rajesh Parekh. Ten Supplementary Analyses to Improve E-commerce Web Sites. *In Proceedings of the Fifth WEBKDD Workshop: Webmining as a premise to effective and intelligent Web Applications*, ACM SIGKDD, Washington, DC, USA, August 2003.

[40] Charles P. Kollar, John R. R. Leavitt, Michael Mauldin, Robot Exclusion Standard Revisited. http://www.kollar.com/robots.html

[41] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon and John Riedl. GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM,* 40(3): 77-87, March 1997.

[42] Pat Langley and Herbert A. Simon. Applications of machine learning and rule induction. *Communications of the ACM,* 38(11):54-64, November 1995.

[43] Brian Lavoie and Henrik F. Nielsen. Web Characterization Terminology \& Definitions Sheet. http://www.w3.org/1999/05/WCA-terms/, May 1995.

[44] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St Pierre, David S. Wells, Monica C. Wong-Chan, Shaw-Wen Yang and Robert Zak. The network architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*. 33(2): 145-158, March 1996.

[45] Sanjay Kumar Madria, Sourav S. Bhowmick, Wee Keong Ng and Ee-Peng Lim. Research Issues in Web Data Mining. In *Data Warehousing and Knowledge Discovery,* Florence, Italy, pages 303-312, August 1999.

[46] Bamshad Mobasher, Robert Cooley, and Jaideeo Sricastava. Automatic Personalization based on web usage mining. *Communications of the ACM*, 43(8):143 –151, August 2000.

[47] Bamshad Mobasher, Namit Jain, Eui-Hong Han and Jaideep Srivastava. *Web mining: Pattern discovery from world wide web transactions.* Technical Report TR-96050, Department of Computer Science, University of Minnesota, Minneapolis, September 1996.

[48] Scott Oaks and Henry Wong. *Java Threads.* O'Reilly, Cambridge, MA, September 2004.

[49] Gregory Piatetsky-Shapiro, Ron Brachman, Tom Khabaza, Willi Kloesgen and Evangelos Simoudis. An Overview of Issues in Developing Industrial Data Mining and Knowledge Discovery Applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining,* Minneapolis, Minnesota, pages 89-95, September 1996.

[50] James Pitkow. In Search of Reliable Usage Data on the WWW. In *Proceedings of the Sixth International WWW Conference,* Santa Clara, California, pages:451-463, April 1997.

[51] Iko Pramudiono, Takahiko Shintani, Katsumi Takahashi and Masaru Kitsuregawa. User Behavior Analysis of Location Aware Search Engine. In *Proceedings of the Third International Conference on Mobile Data Management,* Singapore, pages 139-145, January 2002.

[52] Technical Publications, SGI. SGI - C Language Reference Manual. June 2003.

[53] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan and John Riedl. Getting To Know You: Learning New User Preferences in Recommender Systems. In *Proceedings of the 7th International Conference on Intelligent User Interfaces,* San Francisco, California, pages 127-134, January 2002. ACM Press.

[54] Kendall Square Research. KSR-1 Principles of Operation. Waltham, MA, October 1991.

[55] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computre Supported Cooperative Work*, Chapel Hill, North Carolina, pages 175 - 186, October 1994. ACM.

[56] J. Ben Schafer, Joseph A. Konstan and John Riedi. Recommender systems in E-commerce. In *ACM Conference on Electronic Commerce,* Denver, CO, USA, pages 158 - 166, November 1999.

[57] Cyrus Shahabi, Amir M. Zarkesh, Jafar Adibi and Vishal Shah. Knowledge Discovery from Users Web-Page Navigation. *In Proceedings of the IEEE Seventh International Workshop on Research Issues in Data Engineering,* Birmingham, England, pages 20-29, April 1997.

[58] Upendra Shardanand and Patti Mase. Social information filtering: Algorithms for automating "word of mouth". In *Proceeding of ACM Computer Human Interaction CHI'95 Conference on Human Factors in Computing Systems*, Denver, Colorado, pages 210 – 217, May 1995.

[59] Myra Spiliopoulou, Bamshad Mobasher, Bettina Berendt and Miki Nakagawa. A Framework for the Evaluation of Session Reconstruction Heuristics in Web Usage Analysis. *INFORMS Journal on Computing,* 15(2):171-190, Spring 2003.

[60] Takayuki Tamura, Masato Oguchi and Masaru Kitsuregawa. Parallel Database Processing on a 100 Node PC Cluster: cases for decision support query processing and data mining. In *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing,* pages 1-16, San Jose, California, November 1997. ACM Press.

[61] Pang-Ning Tan and Vipin Kumar. Discovery of Web Robot Sessions Based on Their Navigational Patterns. *Data Mining and Knowledge Discovery,* 6:9-35, January 2002.

[62] Guy Tremblay. Threaded-C Release 2.0: Motivation, Description, and Rationale. CAPSL, Computer Architecture and Parallel Systems Laboratory, Technical Note 09. University of Delaware, Newark, DE, USA. June 2000.

[63] Osmar R. Zaiane, Man Xin and Jiawei Han. Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs. In *Proceeding of Advances in Digital Libraries*, Santa Barbara, California, pages 19 - 29, April 1998

[64] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New Algorithms for fast Discovery of Association Rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD97)*, pages 283-286, California, USA. August 1997.

[65] Albert Y. H. Zomaya. *Parallel and distributed computing handbook.* McGraw-Hill, New York, January 1996.