International Journal of Computer Science and Applications,
©Technomathematics Research Foundation
Vol. 14, No. 2, pp. 115 – 129, 2017

A MAPREDUCE-BASED PARALLEL DATA CLEANING ALGORITHM IN WEB USAGE MINING

MITALI SRIVASTAVA

Department of Computer Science, Institute of Science
Banaras Hindu University, Varanasi – India
mitali.srivastava2011@gmail.com

RAKHI GARG

Computer Science Section, Mahila Maha Vidyalaya
Banaras Hindu University, Varanasi – India
rgarg@bhu.ac.in

P. K. MISHRA

Department of Computer Science, Institute of Science
Banaras Hindu University, Varanasi – India
mishra@bhu.ac.in

Data cleaning plays an important role in Web usage mining process as it reduces the complexity of later phases of KDD (Knowledge Discovery in Databases) process. It incorporates the removal of those log entries and fields that are not relevant for the purpose of Web usage mining. Generally, Data cleaning includes the removal of unsuccessful requests, irrelevant files' requests, inappropriate access methods' requests, and robots' requests. With the enormous increase of log data, traditional Data cleaning algorithm has become highly data intensive task. In the last few years, MapReduce has emerged as most widely used parallel programming framework to compute data intensive application on a cluster of nodes. In this article, we have proposed a MapReduce-based Parallel Data cleaning (PDC) algorithm that considers all aspect of the traditional Data cleaning algorithm. There are two modifications to exiting Data cleaning algorithm *i.e.* first is an efficient robot detection approach, and other is consideration of internal dummy connections' requests .We have performed experiments on real Web server log and observed that log file is reduced to significant amount after considering these two modifications. Moreover, we have compared parallel data algorithm with its sequential version and found that our proposed MapReduce-based PDC algorithm is scalable and efficient than traditional Data cleaning algorithm for larger datasets.

Keywords: Data Cleaning; Data Preprocessing; Web Usage Mining; Web Mining; MapReduce; Hadoop; Web Server Log.

1. Introduction

Web usage mining is one of the important application of data mining that is used to extract patterns from data collected as a result of a users' interaction with one or more Web sites. The ultimate goal of Web usage mining is to extract patterns about profiles and navigation behavior of users [Liu, (2007)]. These patterns can be further used in a variety of applications such as restructuring Websites, improving navigation by prefetching and

caching activities, personalizing Web contents and business intelligence [Facca & Lanzi, (2005)].

Web usage mining process follows typical Knowledge Discovery in Databases (KDD) process and is categorized into three main inter-dependent phases: Data Preprocessing, Pattern discovery, and Pattern analysis [Cooley *et al.*, (2000)]. In Data preprocessing phase, log data are collected from one or more sources, cleaned and prepared in a suitable format so that a particular data mining algorithm can be applied efficiently. In Pattern discovery phase, patterns are extracted by using basic data mining algorithms such as Association rule mining, Classification, Clustering, Sequential rule mining etc. In Pattern analysis phase, extracted patterns are filtered and processed by using domain knowledge and content structure of a Web site [Pabarskaite & Raudys, (2007)]. Due to the unstructured and vast amount of log data, Data preprocessing has become a complex and time-consuming phase in the Web usage mining process. Data preprocessing step is necessary to ensure scalability and efficiency for the next phases of Web usage mining [Pierrakos *et al.*, (2003)]. Data preprocessing of server log incorporates several steps: Data fusion, Data cleaning, User identification, Session identification, Path completion, and Data formatting. Among them, some steps are taken from the process of data preparation of KDD but few like User identification, Session identification etc. are still unique due to the unstructured nature of log data.

In Data fusion phase, log files are collected from multiple Web servers and merged into a single file and synchronized globally [Tanasa & Trousse, (2004)]. Data cleaning refers to the removal of irrelevant entries from log file that are not relevant for Web usage mining process. Data cleaning also incorporates dropping of log file attributes that are not required for next phases of mining task. In User identification phase, a unique user is identified and its activities are grouped and written into a user activity file. In Session identification phase, activities of each user are segmented into multiple sessions. Path completion phase tries to find out missing log entries that are not recorded into log file due to use of back button or caching of proxy server. In Data formatting phase, session or transaction data are formatted according to a particular data mining algorithm *e.g.* temporal information is not required for Association rule mining algorithm hence final preprocessed module would remove time [Castellano *et al*, (2007); Cooley *et al.*, (2000); Liu, (2007)].

In this paper, we have focused on Data cleaning phase and have designed a parallel Data cleaning algorithm that not only cleans the log file but also scales efficiently for the large size of log data. This article is divided into following sections: Section II comprises problem formulation of Data cleaning. Section III covers motivation and literature review. Section IV discusses the basics of MapReduce Programming framework. Section V discusses proposed methodology for Data cleaning algorithm. Section VI includes experiments and analysis of our work. The last section VII concludes the paper.

2. Problem Formulation

Whenever a user requests a resource from a Web site, an entry is logged into a log file called server log. Several formats exist to represent an entry in server log *e.g.* IIS

standard/Extended, Netscape Flexible, NCSA Common/Combined etc. Among them, NCSA extended common log format (ECLF) is widely used customized log format [Pabarskaite & Raudys, (2007); Srivastava *et al.*, (2015)]. Data cleaning problem formulation for ECLF log is given below:-

Suppose $IP = \{ip_1,\ ip_2, \dots, ip_{n_{ip}}\}$ is set of all IP addresses of users who accessed the Web site, $R = \{r_1,\ r_2, \dots, r_{n_R}\}$ denotes the set of all Web resources of the Web site, $U = \{ua_1,\ ua_2, \dots, ua_{n_U}\}$ is set of all user agents of Web user, and $X = \{x_1,\ x_2, \dots, x_{n_X}\}$ is set of links outside the Web site. A log entry in ECLF can be defined as $l_i = < ip_i;\ t; m;\ r_i;\ v; s; b; [ref_i]; [ua_i]; [cookies] >$. Where $ip_i \in IP, r_i \in R, ref_i \in R \cup X, ua_i \in U$, t denotes access time of request *i.e.* Timestamp, m denotes mode of request *i.e. HTTP access method*, v denotes *version of HTTP*, s denotes *HTTP status code*, and b denotes *Number of bytes transferred*. ref_i,ua_i , and $cookies$ are customized attributes. Web server log comprises $L = \{l_1,\ l_2, \dots, l_{n_L}\}$. So we can formalize the Data cleaning problem as follows: For a given Web server log *L*, extract cleaned log $CL = \{cl_1, cl_2, \dots, cl_{n_{CL}}\}$ contains relevant entries and $l_i = < ip_i;\ t; r_i;\ b; [ref_i]; [ua_i]; [cookies] >$. Sometimes *Cookies* are disabled by Web server and *Number of bytes transferred* attribute is not required for few applications then $l_i = < ip_i;\ t; r_i; [ref_i]; [ua_i] >$

3. Motivation and Literature Review

Data preprocessing takes almost 80% time of the whole Web usage mining process and Data cleaning is one of an important phase of it. The log file contains plenty of information that is not relevant for analyzing users' navigation behavior [Pabarskaite, (2002)]. This raw information increases the size of log file that becomes overhead for the next phases of Web usage mining process. Data cleaning has become a data intensive task due to the huge size of log files. The whole data preprocessing task can be fastened up by decreasing the time taken by the Data cleaning phase. In the last few years, MapReduce has become one of the prevalent and widely used parallel programming framework for processing large datasets on a cluster of nodes [Cardosa *et al.*, (2011)]. Hadoop is an open source software framework incorporated MapReduce programming approach that provides scalable, flexible, fault tolerance, distributed and data intensive computing over the cluster of computers [Karim *et al.*, (2013); Lin & Dyer, (2010)].

In Recent years, several research works have been carried out in the field of Data preprocessing in Web usage mining. [Cooley *et al.*, (2000)] have proposed several data preprocessing techniques such as Data cleaning, User identification, Session identification etc. For Data cleaning, they have considered the requests of filename suffixes such as jpeg, jpg, gif, map, and cgi. For robot detection, they have considered robots.txt check, matching *User agent* with the list of known robots. [Tanasa & Trousse, (2004)] have proposed the advanced data preprocessing technique which is categorized into four main steps: Data fusion, Data cleaning, Data structuration and Data summarization. They have classified Data cleaning into two steps: One is removing requests for non-analyzed resources such as graphics and maps. Another one is requests generated by Web robots. For detecting robots' request, they have applied robots.txt check, matching *User agent* with the list of known

robots and calculating browsing speed methods. However, unsuccessful requests are not eliminated from the log file. [Castellano et al., (2007)] have developed LODAP (Log data Pre-processor) tool to perform data preprocessing. They have done the cleaning by removing unsuccessful requests, multimedia and script files' requests, requests that contain *HTTP access method* except for GET and requests generated by robots. Robots are identified by robots.txt check and by calculating browsing speed methods. However, all above methods are efficient but they fail at scalability points. Few researchers have focused on server log analysis using MapReduce approach. [Zhang & Zhang, (2013)] have done Data Preprocessing using MapReduce framework. In map task, they have cleaned requests of image, style sheet, and script file, requests having *HTTP status code* less than 200 and greater than 299 and requests having *HTTP access method* except GET. In map phase, output key is an *IP address* and value is a combination of *Requested url*, *Referrer url*, and *Timestamp*. In reducer phase, they have performed other preprocessing task that is Session identification. However, they have not applied any robot detection methods for cleaning. [Savitha & Vijaya, (2014)] have analyzed server log using the MapReduce programming approach for statistical analysis. In map task for Data cleaning, they have considered *HTTP status code* except 200, *HTTP access method* except for GET and robots.txt check. They have selected *Timestamp*, *date*, and *byte transfer* and *browser version* fields for further processing. They have set reduce task to zero, which results in multiple cleaned log file based on a number of mappers created.

In this paper, we have considered the requests of multimedia files, style sheet file, script files, flash animation file, requests having *HTTP status code* less than 200 and greater than 299, requests containing *HTTP access method* except for GET and POST, requests generated by robots. Robots are identified by using method *i.e.* robots.txt check and heuristic by matching keywords like bot, spider, indexer and crawler in *User agent* string. We have taken one more attribute for Data cleaning *i.e.* requests by internal dummy connection. We have implemented both sequential and MapReduce-based PDC algorithm and observed the result in detail.

4. MapReduce Programming Framework

MapReduce is a programming framework for implementing distributed computation for processing data on terabyte and petabyte scale in a reliable and fault tolerant manner. Its programming model mainly works with two functions i.e. Map and Reduce that are expressed mathematically by the equation (1) and (2) [Dean & Ghemawat, (2008)]:

$$map\ (key1, value1) \rightarrow list(key2, value2); \quad (1)$$

$$reduce\ (key2, list\ [value2]) \rightarrow list(key3, value3); \quad (2)$$

When a particular MapReduce job is allocated to the cluster system for execution, multiple map tasks are started on various nodes. In each map task, a map function is applied to each key- value pair that is associated with it and generates a set of intermediate key-value pairs. These intermediate results are sorted by keys and stored in local file system. After that MapReduce engine informs reducer to start its execution. Reducer merges all

intermediate values associated with the same intermediate key and generates output key value pairs [Lin & Dyer, (2010)]. Hadoop is widely used open source implementation framework for MapReduce approach. Hadoop provides two layers for implementation: Data storage engine (HDFS) and Processing engine (MapReduce Framework) [Bhandarkar, (2010)]. HDFS is a block -structured, distributed file system which stores large scale data across the nodes in the cluster. HDFS follows master-slave architecture. There are mainly two types of nodes that are created on HDFS called Name node and Data node [1]. Name node and Data node are also called as Master server and Slave node respectively. Name node manages file system namespace i.e. hierarchy of files and directories and executes file system operations such as opening, renaming, deleting and closing of files and directories. It also maintains the mapping of blocks to Data nodes. There are several Data nodes corresponding to a name node in the cluster. Data nodes are responsible for serving read /write requests of blocks to file system clients. These nodes also store and manage the blocks assigned to them. Input raw log should be loaded into HDFS to start processing by MapReduce job. At the time of loading, the input file is partitioned into fixed-sized blocks and each block is replicated to Data nodes according to replication factor to ensure fault tolerance.

Hadoop processing engine consists of two specific functions: Job tracker and Task tracker. Job tracker runs on the Name node of MapReduce framework and implements scheduler module that is responsible for assigning the MapReduce tasks to Data nodes. Job tracker is also responsible for monitoring, reassigning the task when it fails. On the other hand, each Data node runs a task tracker to accept the task from Job tracker. Whenever a particular MapReduce job submitted for execution on Hadoop cluster, Job tracker splits job into mappers and reducers and handover them to available task trackers for processing. Further, task tracker accepts the request from job tracker. If the request is a map task then it processes the data block by map specification and if the request is a reduce task then it will wait until receiving a notification from Job tracker (when all mappers are completed execution) to start its execution [F. Li *et al.*, (2014)].

5. Methodology

The process of proposed Data cleaning algorithm is depicted in Fig. 1. Initially, input log file is split into chunks and each chunk is assigned to a mapper to apply Data cleaning algorithm on it. After the completion of all map task, cleaned log chunks are shuffled and sorted according to the key of map task and handover to reduce task. Further, reduce task write all entries into a cleaned log file.

5.1. Data Cleaning

Data cleaning is an initial, domain dependent and important step in data preprocessing phase of Web usage mining. We have considered the following types of records which should be removed from Web server log.

5.1.1. Unsuccessful Requests

HTTP status code attribute specifies the status of users' request whether it is successfully completed or not. Unsuccessful requests are recorded into log file if a bad request is done by user or server is unable to complete this request. For example, requests having *HTTP status code* 400 or 404 are caused by client errors or bad requests [Li *et al.*, (2014)].

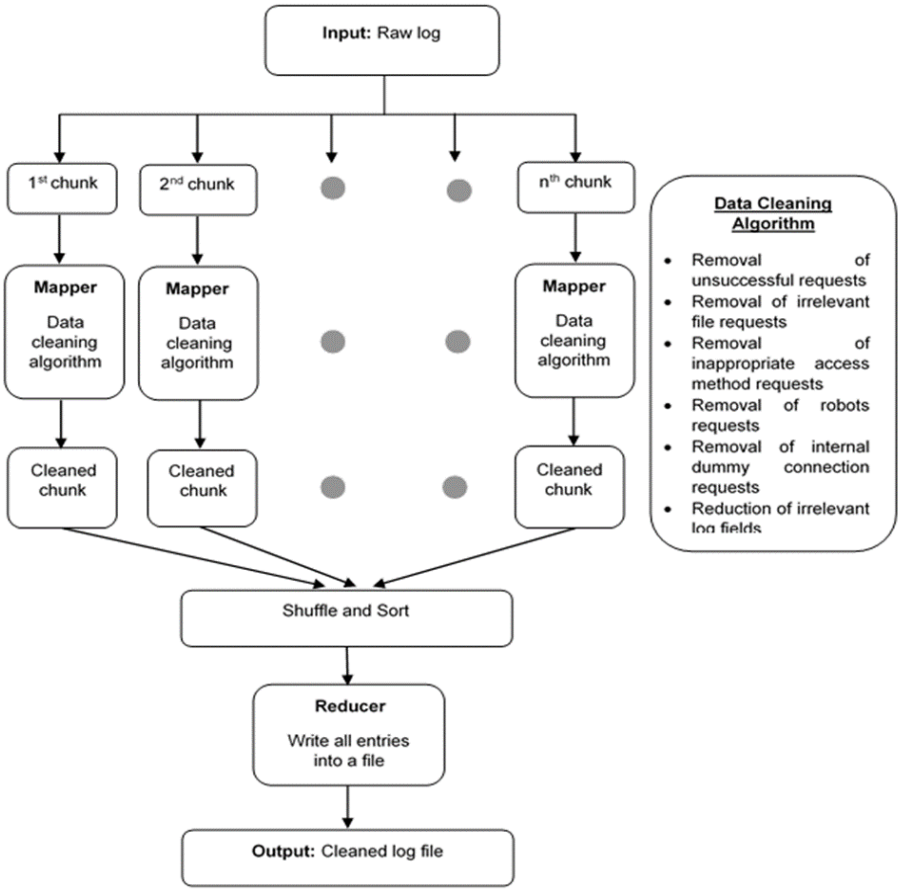

Fig. 1. Methodology of Proposed work

There are five classes defined for *HTTP status code*: Informational (1xx), Success (2xx), Redirect (3xx), Failure (4xx) and Server error (5xx) [Srivastava *et al.*, (2015)]. We have considered only success series [200-299] to put into the cleaned log file.

5.1.2. Irrelevant Files' Requests

Most of the Web pages contain embedded objects like image files, script files, flash files along with actual Web page. Whenever a user requests a page on a Web site, embedded

objects of that page are also requested implicitly and entries are recorded into the server log file [Cooley *et al.*, (2000)]. This is because HTTP protocol needs a separate connection for every file on Web server. In general, log entry of actually requested page should be kept in cleaned log file for analysis and others should be discarded as they do not depict the actual behaviour of the user. When there is a requirement to do the analysis of server activities like prefetching and caching then entries corresponding to these files should be kept in cleaned log file but for another purpose of Web usage mining, these should be removed [Tanasa & Trousse, (2004)]. Requests of embedded objects can be easily removed from log file by checking suffix of *Requested url* field. We have considered multimedia files, style sheet file, flash animation file and script files for removal.

5.1.3. Inappropriate Access Methods' Requests

Requests with *HTTP access method* other than GET or POST are often related to administrator activities, CGI accesses, visits of robots, Web site developer, and properties of the Server [Castellano *et al.*, (2007)]. Since these requests do not belong to a particular user, these should be removed from the log file. We have kept GET and POST containing requests in the cleaned log file as they are related to the user.

5.1.4. Robots' Requests

Web robots also called as Web spiders, Web crawlers, Web bots or Web wanderers, are special type of software that traverses the Web site to extract its contents. Web robots follow hyperlink structure of a Web site to update search engines' index databases [Tan & Kumar, (2004)]. Removing Web robots requests from the log file does not only reduce the size of log file but also remove uninteresting sessions. To identify robots' requests we have applied following heuristic methods:

- Check the requests of robots.txt file [Castellano et al., (2007)].
- Check whether *User agent* contains spider, bot, indexer or crawler keyword [Geens *et al.*, (2006)].

5.1.5. Requests by Internal Dummy Connection

Sometimes server wakes up its child processes that are listening for a new connection. To do this server sends an HTTP request back to itself *i.e.* Apache server wakes up its child processes by sending a request for the root node of the Web site. This request appears in Apache access log with *IP addresses* of localhost such as 127.0.0.1 or 127.1.1.1 for IPv4 or ::1 for IPv6 and *User agent* as internal dummy connection[2]. Since these type of requests belong to a server activity, these should be removed from the log file. The Sample of one entry of log files is given in Fig. 2:

```
:: 1 - - [24/Mar/2014:00:02:15 +0530] "GET / HTTP/1.0" 200 17090 "-" "Apache/2.2.3 (Red Hat) (internal dummy connection)"
```

Fig. 2. An entry in Apache server log by internal dummy connection.

5.1.6. Irrelevant log fields

The log file contains various fields which are not required for the purpose of analysis *e.g. HTTP version, HTTP status code* etc. [Li et al., (2014)]. The log fields such as *IP address*, *User agent* and *Cookies* are required for User identification phase. The *Timestamp* and *Referrer url* are required to apply Session identification. *Requested url* is an important field in Pattern discovery phase of Web usage mining process. *Cookies* are disabled by Web server so only *IP address*, *Timestamp*, *Requested url*, *Referrer url*, and *User agent* are kept in the cleaned log file for the next phases of data preprocessing.

5.2. Proposed Parallel Data Cleaning (PDC) Algorithm

In MapReduce programming approach, input data is a collection of key-value pairs. Initially, raw log file is loaded into HDFS to provide input to MapReduce program. At the time of loading, it is split into multiple blocks according to block size and replicated to Data nodes according to replication factor. There are several types of *FileInputFormat* are specified in Hadoop MapReduce programming framework: *TextInputFormat, NlineInputFormat, KeyValueInputFormat*[3] etc. In our algorithm, we have chosen *NlineInputFormat*. This format splits N lines of input as one split. Here input to the map function *i.e.* (k,v) is (LongWritable, Text), where the key is byte offset of line and value is line contents. The Map and Reduce part of the algorithm are given in Algorithm 1.

5.2.1. Map Phase

The specification of map phase named as CleanMapper is given in Algorithm 1.In map function, for each line *IP address*, *HTTP status code*, *HTTP access method*, *User agent*, and *Requested url* are extracted by parsing the value. In step 3, these attributes are checked with required Data cleaning criteria. If this value is found irrelevant then it is discarded otherwise extract fields *Timestamp*, *Referrer url* by parsing the value (step 4).

Further, map task will generate the intermediate key as *Timestamp* and value as a combination of *IP address, Timestamp, Requested url, Referrer url, User agent*. The output of each map task is a list of (Timestamp, value). After execution of map task, all values are shuffled and sorted according to *Timestamp* implicitly before the reducer starts executing itself.

5.2.2. Reducer Phase

The specification of reduce phase named as CleanReducer is given in Algorithm1. Reducer receives values for each *Timestamp* in sorted order and these values are written into cleaned log.

CleanMapper (key, value)
ip: IP address field
timestamp: Timestamp field
method: HTTP access method field
url: Request url field
status: HTTP status code field
ref: Referrer url field
user_agent: User agent field
file_ext: File extension of Requested url field
Input: *(key: offset in bytes; value: text of a line)*
Output: *(key': timestamp, value': combination of ip, timestamp, url, ref, user_agent)*

```
for each value
    extract status, method, url, ip, user_agent, by parsing value
        if(status ∈ [200-299]) and (method ∈ [GET, POST]) and (file_ext ∉ [jpeg, jpg, png, gif, tif,
        tiff, bmp, ico, css, js, cgi, swf, mp3]) and (ip ∉ [::1, 127.1.1.1, 127.0.0.1]) and (url does not
        contain robots.txt) and (user_agent does not contain keywords [bot, spider, crawler,
        indexer])
        extract timestamp, ref by parsing value
        key': timestamp
        value': combination of ip,timestamp, url, ref ,user_agent
        EmitIntermediate (key', value')
        endif
endfor
```

Algorithm 1. Mapper of PDC algorithm.

CleanReducer (key, values)
Input: (*key: timestamp, values: Entries corresponding to particular timestamp)*
Output: *(key': one value from values, value': NullWritable)*

```
1. for each value in values
2.     emit (key', value')
3. end for
```

Algorithm 1. Reducer of PDC algorithm.

6. Experiment and Result Analysis

We have done experiments to observe the performance of PDC algorithm. Algorithm 1 is implemented in JAVA (JDK1.7) language using Hadoop 2.6.0[3] framework API and is executed on a cluster of 4 nodes, 3 nodes, and 2 nodes respectively. In these cluster setup, one node is master node and rest are data/slave nodes having 4 cores in each node. All nodes have Ubuntu 14.04 operating system. A sequential Data cleaning algorithm is also implemented in JAVA (JDK1.7) and executed on single node to analyse the *Speedup* of PDC algorithm. Algorithm1 is evaluated based on two aspects: first is relevancy of algorithm and second is scalability of the algorithm.

6.1. Dataset Description

For the experiment, we have collected the real Web server log files from Banaras Hindu University Web server *i.e.* Apache/2.2.3 (Red Hat Further, datasets are extracted of sizes D1 (0.72GB), D2 (1.44GB), D3 (2.88GB), and D4 (5.76GB) for analysis. This log is in Extended common log format. From Table 1 it can be observed that datasets are taken in increasing size order. This is because *Sizeup* metric of parallel algorithm can be calculated if datasets are increased by a multiplying factor. Here multiplying factor is 2.

Table 1. Description of Datasets

Datasets	**Start Time**	**End Time**	**Size(GB)**	**Total Number of Requests**
D1(1X)	02/03/2014:07:48:34	05/03/2014:19:43:00	0.72	3361728
D2(2X)	02/03/2014:07:48:34	09/03/2014:08:27:18	1.44	6725269
D3(4X)	02/03/2014:07:48:34	16/03/2014:07:58:10	2.88	13282787
D4(8X)	02/03/2014:07:48:34	30/03/2014:07:58:10	5.76	26057551

6.2. Relevancy of Data Cleaning Algorithm

Table 2 shows the proportion of irrelevant files' requests, unsuccessful requests, inappropriate access methods' requests, robots' requests by robots.txt approach and robots' requests by keywords matching, and requests by internal dummy connection in datasets D1, D2, D3, and D4 respectively. From Table 2, it can be observed that the contribution of irrelevant files' requests is highest among all parameters. Robots' requests identified by robots.txt approach are 0.03% for D1, D2 and 0.04% for D3, D4. The contribution of robots' requests by using keywords matching approach are 0.86%, 0.89%,0.81%, and 0.76% for D1, D2, D3, and D4 respectively. The contribution of internal dummy connections' requests are 1.14% for D1, D3, D4 and 1.15% for D2 datasets.

The combined contribution of all irrelevant parameters without considering internal dummy connection and robots by keywords matching approach *i.e.* existing approach are 78.03%, 77.88%, 78.10% and 78.81% for D1, D2, D3 and D4 respectively.

After applying robots by keywords approach, contribution of all irrelevant parameters becomes 78.51 %, 78.42%, 78.58%, 79.25% for datasets D1, D2, D3, and D4 respectively and log file reduces 0.48%, 0.54%, 0.48%, and 0.44% more for datasets D1, D2, D3, and D4 respectively. After considering internal dummy connections' requests, combined contribution of all irrelevant parameters becomes 79.17%, 79.03%, 79.24%, 79.94% for datasets D1, D2, D3, and D4 respectively and log file reduces almost 1.14% more than existing algorithm for each dataset. The combined contribution of all attributes after both the modifications are 79.65%, 79.57%, 79.72% and 80.38% for D1, D2, D3 and D4 respectively. Our approach reduces 1.62%, 1.69%, 1.62%, and 1.57% more than existing MapReduce-based Data cleaning algorithm for datasets D1, D2, D3, and D4 respectively. Only 20.35%, 20.43%, 20.28% and 19.62% entries are found relevant for D1, D2, D3, and D4 datasets for next phases of Web Usage Mining process. This shows the effectiveness of PDC algorithm.

Table 2. Proportion of various irrelevant parameters in log file

Various Attributes (%) \ Datasets	D1	D2	D3	D4
Irrelevant files' requests	76.80	76.66	76.94	77.72
Unsuccessful requests	19.47	19.36	19.11	19.13
Inappropriate access methods' requests	0.20	0.22	0.22	0.21
Robots' requests by robots.txt	0.03	0.03	0.04	0.04
Robots' requests by keywords matching	0.86	0.89	0.81	0.76
Internal dummy connections' requests	1.14	1.15	1.14	1.14
All except robots requests by keywords and internal dummy connection	78.03	77.88	78.10	78.81
All except robots requests by keywords matching	79.17	79.03	79.24	79.94
All except internal dummy connection requests	78.51	78.42	78.58	79.25
All attributes	79.65	79.57	79.72	80.38

6.3. Scalability of PDC algorithm

To evaluate efficiency and scalability of PDC algorithm, following two metrics Speedup and Sizeup are considered [Xu et al., (1999)].

6.3.1. Analysis of Speedup of PDC Algorithm

Speedup is the most widely used parameter to compare the efficiency of a parallel algorithm with sequential version of algorithm. It measures how much a parallel algorithm is faster than its corresponding sequential algorithm. It is defined as a ratio of time taken by sequential algorithm to parallel one as shown in (Eq. 3).

$$\boldsymbol{Speedup}\ (\boldsymbol{n},\boldsymbol{p}) = \frac{\boldsymbol{T_{sequential}(n)}}{\boldsymbol{T_{parallel}(n,p)}} \qquad (3)$$

Where n denotes data size, p is a number of nodes in a cluster, $T_{sequential}(n)$ represent time taken by sequential algorithm on 1 node and $T_{parallel}(n,p)$ is time taken by parallel algorithm on a cluster with p nodes.

To calculate the *Speedup*, we have kept the data size fixed and increased the number of Data nodes from 1 to 3. The numbers of mappers are chosen as 3, 6, 12 and 24 for datasets D1, D2, D3, and D4 respectively. From Fig. 3, we can observe that *Speedup* increases linearly as Data nodes increases from 1 to 2 for D1, D2, D3, and D4 and closes towards ideal speedup. But on increasing Data nodes from 2 to 3, *Speedup* decreases for datasets D1, slightly increases for datasets D2 and linearly increases for datasets D3 and D4.This is because, for smaller datasets distributing and managing task takes more time

than processing time hence *Speedup* reduces by increasing number of Data nodes. On the other hand for larger datasets, processing time is more than managing time and that could be reduced by the parallelism by increasing number of Data nodes. This shows that MapReduce-based PDC algorithm scales linearly for larger datasets.

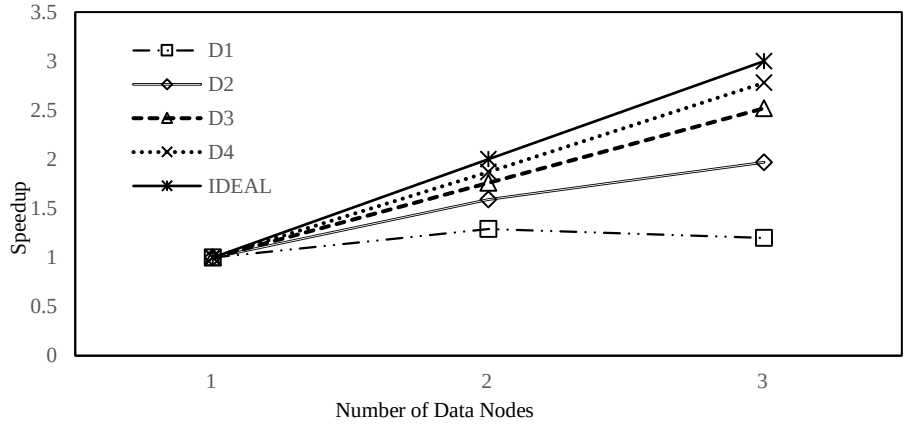

Fig. 3. Speedup of PDC algorithm on D1, D2, D3 and D4 datasets

6.3.2. Analysis of Sizeup of PDC Algorithm

Sizeup metric is used to calculate the effectiveness of a parallel algorithm to handle growth of data size. It measures how much longer time the parallel algorithm takes on a fixed number of nodes when data size is increased by a multiplying factor. *Sizeup* is defined in Eq.4.

$$Sizeup\ (p, x) = \frac{T_{parallel}(p, xn)}{T_{parallel}(p, n)} \tag{4}$$

Where n denotes data size, x denotes multiplying factor to data size, p is a number of nodes in a cluster, $T_{parallel}(p, xn)$ represents the time taken by parallel algorithm for $x \times n$ size dataset on cluster with p nodes and $T_{parallel}(n, p)$ is time taken by parallel algorithm of n -size dataset on same number of nodes.

To evaluate *Sizeup*, we have fixed the number of Data nodes as 1, 2 and 3 and increased dataset D1 by factor of 2, 4 and 8. From Fig. 4, it can be observed that when data size increases by multiplying factor x Sizeup does not increase according to proportion of x. For example when size is 8 times of D1 the *Sizeup* values are 6.67, 5.12 and 3.18 on 1 Data node, 2 Data nodes and 3 Data nodes respectively. These *Sizeup* values are less than 8 and decreases with the increase of Data nodes that shows that PDC algorithm gives excellent *Sizeup* for larger datasets

From these observations we can conclude that proposed PDC algorithm provide more accurate cleaned data, linear *Speedup* and excellent *Sizeup*.

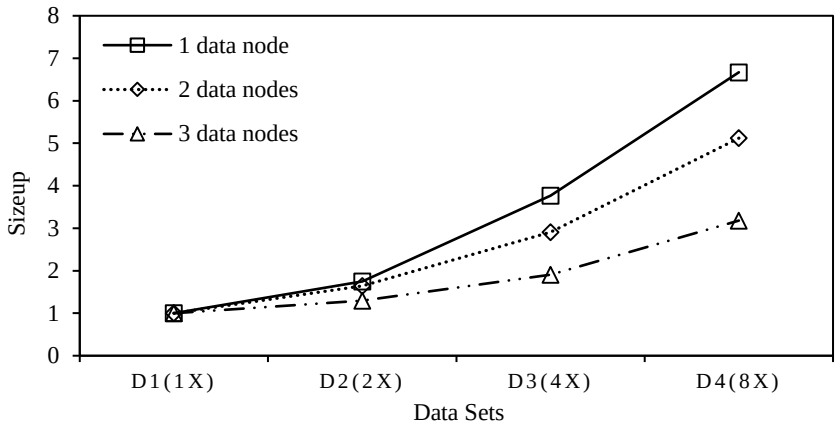

Fig. 4. Sizeup of PDC algorithm on D1, D2, D3, and D4 datasets

6.4. Comparison of Our Proposed Work with Previous Work

In this section, we have compared our approach with previous work in Data cleaning based on parameters such as unsuccessful requests, irrelevant files' requests, inappropriate access methods' requests, robots' requests, internal dummy connections' requests, and scalability. Table 3 shows the comparison of our work with the existing one where the value "Yes" and "-" shows the consideration and non-consideration of particular cleaning parameters. Here, scalability means consideration of parallel approach and ability to deal with large datasets. From Table 3 it can be observed that, our proposed Data cleaning algorithm considers maximum parameters for Data cleaning those are not considered together by previous researchers.

Table 3: Comparison of our approach with previous work in Data cleaning

Author	**Unsuccessful requests**	**Irrelevant files' requests**	**Inappropriate access methods' requests**	**Robots' requests**		**Internal dummy connections' requests**	**Scalability**
				robots.txt	**Other methods**		
Cooley et al.	-	Yes	-	Yes	Yes	-	-
Pabarskaite Z.	-	Yes	-	Yes	-	-	-
Huiying, Z	-	Yes	-	-	-	-	-
Tanasa et al.	-	Yes	-	Yes	Yes	-	-
Chaofeng, L.	Yes	Yes	-	Yes	-	-	-
Castellano et al.	Yes	Yes	Yes	Yes	Yes	-	-
Aye T. T.	Yes	Yes	Yes	-	-	-	-
Reddy et al.	Yes	Yes	-	-	-	-	-
Losarwar et al.	Yes	Yes	Yes	Yes	-	-	-
Vellingiri,J	Yes	Yes	Yes	Yes	Yes	-	-
Zhang et al.	Yes	Yes	Yes	-	-	-	Yes
Savitha et al.	Yes	-	Yes	Yes	-	-	Yes
Our Method	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

7. Conclusion

Due to huge amount of log data, existing Data cleaning algorithm faces scalability and efficiency issues. In this article, we have proposed a MapReduce-based parallel Data cleaning algorithm (PDC) that includes all features of traditional Data cleaning algorithm. Moreover, it also incorporates one more attribute for Data cleaning of server log *i.e.* requests by internal dummy connection. By experiment, it is shown that after taking this attribute, log file size reduces to almost 1.14% more than by traditional Data cleaning algorithm. For robot detection an efficient heuristic is applied that removes the significant amount of robots requests. To measure scalability, we have calculated Speedup and Sizeup metric for proposed algorithm and it provides linear Speedup and excellent Sizeup for larger datasets. The output of PDC algorithm can be further used for other phases of data preprocessing such as user identification and session identification. In future, PDC algorithm can be enhanced by applying other potential robot detection techniques.

Notes

1. HDFS architecture guide, https://hadoop.apache.org/docs/r1.0.4/hdfs_design.pdf.
2. Internal Dummy connection https://wiki.apache.org/httpd/InternalDummyConnection.
3. Hadoop Map/Reduce Tutorial. ApacheTM, 2014, http://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

References

Aye, T. T. (2011). Web log cleaning for mining of web usage patterns. Paper presented at the Computer Research and Development (ICCRD), 2011 3rd International Conference on.

Bhandarkar, M. (2010). MapReduce programming with apache Hadoop. Paper presented at the Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on.

Cardosa, M., Wang, C., Nangia, A., Chandra, A., & Weissman, J. (2011). Exploring mapreduce efficiency with highly-distributed data. Paper presented at the Proceedings of the second international workshop on MapReduce and its applications.

Castellano, G., Fanelli, A., & Torsello, M. (2007). LODAP: a log data preprocessor for mining web browsing patterns. Paper presented at the Proceedings of the 6th Conference on 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases.

Chaofeng, L. (2006). Research and development of data preprocessing in Web Usage Mining. In International Conference on Management Science and Engineering.

Cooley, R. W., & Srivastava, J. (2000). Web usage mining: discovery and application of interesting patterns from web data. Minneapolis, MN: University of Minnesota.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

Facca, F. M., & Lanzi, P. L. (2005). Mining interesting knowledge from weblogs: a survey. Data & Knowledge Engineering, 53(3), 225-241.

Geens, N., Huysmans, J. and Vanthienen, J. (2006). Evaluation of web robot discovery techniques: a benchmarking study. Industrial Conference on Data Mining, pp. 121-130.

Huiying, Z., & Wei, L. (2004, June). An intelligent algorithm of data pre-processing in Web usage mining. In Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on (Vol. 4, pp. 3119-3123). IEEE.

Karim, M. R., Ahmed, C. F., Jeong, B.-S., & Choi, H.-J. (2013). An efficient distributed programming model for mining useful patterns in big datasets. IETE Technical Review, 30(1), 53-63.

Li, F., Ooi, B. C., Özsu, M. T., & Wu, S. (2014). Distributed data management using MapReduce. ACM Computing Surveys (CSUR), 46(3), 31.

Li, M., Yu, X., & Ryu, K. H. (2014). MapReduce-based web mining for prediction of web-user navigation. Journal of Information Science, 0165551514544096.

Lin, J., & Dyer, C. (2010). Data-intensive text processing with MapReduce. Synthesis Lectures on Human Language Technologies, 3(1), 1-177.

Liu, B. (2007). Web data mining: exploring hyperlinks, contents, and usage data: Springer Science & Business Media.

Losarwar, V., & Joshi, D. M. (2012). Data Preprocessing in Web Usage Mining. Paper presented at the International Conference on Artificial Intelligence and Embedded Systems (ICAIES'2012) July.

Pabarskaite, Z. (2002). Implementing advanced cleaning and end-user interpretability technologies in web log mining. Information Technology Interfaces ITI2002, Collaboration and Interaction in Knowledge-Based Environments.

Pabarskaite, Z., & Raudys, A. (2007). A process of knowledge discovery from web log data: Systematization and critical review. Journal of Intelligent Information Systems, 28(1), 79-104.

Pierrakos, D., Paliouras, G., Papatheodorou, C., & Spyropoulos, C. D. (2003). Web usage mining as a tool for personalization: A survey. User modeling and user-adapted interaction, 13(4), 311-372.

Reddy, K. S., Reddy, M. K., & Sitaramulu, V. (2013). An effective data preprocessing method for Web Usage Mining. Paper presented at the Information Communication and Embedded Systems (ICICES), 2013 International Conference on.

Savitha, K., & Vijaya, M. (2014). Mining of web server logs in a distributed cluster using big data technologies. International Journal of Advanced Computer Science and Applications (IJACSA), 5(1).

Srivastava, M., Garg, R., & Mishra, P. (2015). Analysis of Data Extraction and Data Cleaning in Web Usage Mining. Paper presented at the Proceedings of the 2015 International Conference on Advanced Research in Computer Science Engineering & Technology (ICARCSET 2015).

Tan, P.-N., & Kumar, V. (2004). Discovery of web robot sessions based on their navigational patterns Intelligent Technologies for Information Analysis (pp. 193-222): Springer.

Tanasa, D., & Trousse, B. (2004). Advanced data preprocessing for intersites web usage mining. IEEE Intelligent Systems, 19(2), 59-65.

Vellingiri, J. (2011, February). Pandian "A Novel Technique for Web Log mining with Better Data Cleaning and Transaction Identification. In Journal of Computer Science.

Xu, X., Jäger, J., and Kriegel, H.-P. (1999). A fast parallel clustering algorithm for large spatial databases. Data Mining Knowledge Discovery. vol. 3, no. 3, pp. 263-290.

Zhang, G., & Zhang, M. (2013). The Algorithm of Data Preprocessing in Web Log Mining Based on Cloud Computing. Paper presented at the 2012 International Conference on Information Technology and Management Science (ICITMS 2012) Proceedings.