TP C++ 3 : Analyse de logs Apache

Edern HAUMONT and Nicolas SIX

B3111

13 décembre 2015

1 Specifications

1.1 General specifications

Our program is designed to deal with apache servers log files.

The input of our program (analog) is a correctly built (cf. detailed specs) log file (.log). The output is the list of the ten most visited URL on the server. This list is printed on the standard output. Some options are available to detail the request. The program is able to generate a .dot file which may be viewed with Softwares like Graphviz.

Our program is made to deal quickly with big log file to be easily updated if the treatment needs of the user evolve. That is, if someone wants to modify the application to add a classification level to the Data structure, it must be easy for him/her. The users should not be a programmer, simply a server administrator.

The program is designed for linux systems. But should work on other platforms.

1.2 Log lines

A log line must fulfill several conditions in order to be accepted by the program. Besides the conditions on each information, there is a general structure to check :

- ☐ there is a specific order for the informations in a logline
- ☐ each information in seperated from the others with a space. However, the request, the referrer, and browser informations are given between double quotes.

Each information of a log line is the object of a small test

Element	Description	Test	Additional informations
Ipv4 adress	The program does not check its validity	TestIpv4	none
User logname	It must be in one world. If there is none, it is replaced by a dash ("-")	TestPseudo&Logname	none
Authenticated User (Pseudo)	It must be in one world. If there is none, it is replaced by a dash ("-")	TestPseudo&Logname	none
Date, time and GMT	As followed : [DD/-Mon/YYYY :HH :MM :SS XGMT] (X replaced by "+" or "-")	TestDate&hour&GMT	These conditions may be added — date < current date — hour between 00 :00 :00 and 23 :59 :59 — GMT between -12 and +12
Total request	As followed (no constraint of world size) : "REQUESTTYPE requestedURL requestProtocol"	TestRequestType, TestRequestDestination, TestRequestProtocol	The only request that must be considered as valid by the program is GET, the other are accepted but not treated
Return code	Between 100 and 400, codes above 300 included are considered as fail codes	TestReturnCode	none
Size of transmission	>= 0. If it is unknown, it is replaced by a dash ("-")	TestDataSize	none
Referrer	It is given between double quotes	TestReferrer	none
Browser and browser infos	Given between double quotes	TestBrowser	No particular syntax

Here is a log line template given as example :
MyIpAdress Logname Pseudo [DD/Mon/YYYY :HH :MM :SS XGMT] "REQUESTTYPE requestedURL requestProtocol" COD SIZE "referrer" "Browser informations given without a specific order"

And an exemple :
192.168.0.0 - - [08/Sep/2012 :11 :16 :02 +0200] "GET /temps/4IF16.html HTTP/1.1" 200 12106 "http ://intranet-if.insa-lyon.fr/temps/4IF15.html" "Mozilla/5.0 (Windows NT 6.1 ; WOW64 ; rv :14.0) Gecko/20100101 Firefox/14.0.1"

1.3 Accepted requests

The program's name is "analog". It can be called with the name of a log file and possibly several options. Even if the options do not change the way a log file is read, it changes the kind of output or, at least a certain selection of treated informations.

Here is the template of a call :

./analog [options] filename.log

Here are three available options :

option	description	test
-e	e stands for exceptions. With this option, the program must exclude of the treatment all documents (in the log requests) that have a picture, css or javascript extension. As it is accepted if the program only exclude most common extensions, we chose to let the user define them in a text file (excludedExtensions.txt)	TestRequestE
-g outputName.dot	g stands for graphic. The program must generate a .dot file, compatible with the software graphviz. In this graph file, the nodes are documents and the arcs are a travelling through a document to another. These arcs correspond to the link between a referrer URL and a destination URL in a log line	TestRequestG
-t hour	t stands for time. hour is a number between 0 and 23. This option is used to select only one hour of data between hour and hout+1. That means, the results in the output and/or those in the dot file are those corresponding to the trafic on the server in this interval of time. It is as if the logfile lines out of this interval didn't exist	TestRequestT
none	If there is no option -g, the program will return the 10 most used resources which fill the options. If there are less than 10 entries, it only prints those that are stored.	TestRequestSimple, TestNoLogfile, TestEmptyLogFile

1.4 Data structure

We want a data structure which stores one information for each referrer-destination-hour combination : the number of hits. It must be fast and easily upgradable, so all informations, including those which are useless must also be stored. The Data structure must be optimized for readings in priority. However, if a future user wants to add an additional sorting level, it must be easy for him/her.

2 Conception and Data Structure

2.1 Data storage

For our Data storage structure, we chose to sort informations by referrer, destination (requestedURL), and hour, in this order.

As there are 24 hours in a day, the sorting by hour is easily implemented by an array of 24 cells. For the two other informations though, the number of possible values depends on the log file. Consequently, we chose to use balanced binary trees as an effective dynamic structure.

You can find at the last page some patterns that resume the Data storage structure at the end of the document.

To resume, we have a main binary tree ordered by destination. Each cell contains other binary trees ordered by referrer. Each cell contains a static array ordered by hour. Each cell contains a vector of other info.

We chose to keep other infos in a class so that, if the user wants to add a level of sorting, he/she just has to take it from this class (as the parser already gets it)

2.2 Classes

We have 3 classes.

You can find at the end of the document the general pattern of the application, with the interactions between classes.

2.2.1 DataManager

His is our main class. It contains our Data structures.

It has two main methods :

int LoadLogFile(const std : :string &logFilePath) ;

This method is used to load a logfile in the program. The parameter logFilePath is the path to the file.

int Request(const bool optionT=false,const int tHour=-1,const bool optionE=false,const bool optionG=false,const std : :string &outputFile="") ;

This method generates the outputs of the program. The parameters correspond to user options, outputFile is the path to a generated .dot file

2.2.2 GraphGenerator

The only usage of this class is its ability to generate a .dot file. This file is created at the construction of the object.

There are two public methods in this class :

int addNodeToGraph(const std : :string &nodeName) ;

This method adds a Node to the graph, the parameter is the name of the Node.

int addLinkToGraph(const std : :string &nodeNameFrom, const std : :string &nodeNameTo, const std : :string &linkLabel) ;

This method adds a link between the two nodes of the graph given in parameters. The link has a label (3rd parameter).

2.2.3 LogOtherInfos

This class contains all infos of a log line that are not used to store informations in our data structure. As the information is already parsed by the DataManager class, if someone wants to

upgrade the application, he/she will only have to take the attribute he/she wants from this class and make it a sorting element.

FIGURE 1 – data structure