Artail et al. Journal of Internet Services and Applications (2017) 8:2
DOI 10.1186/s13174-016-0053-9

Journal of Internet Services and Applications

RESEARCH Open Access

A framework for identifying the linkability between Web servers for enhanced internet computing and E-commerce

Hassan Artail*, Ammar El Halabi, Ali Hachem and Louay Al-Akhrass

Abstract

Existing Web session tracking techniques mainly work on huge web server log files, which limit their dynamicity and capability. We present a collaborative method for session tracking to identify direct and indirect web server interactions. The outcome could lead to building partnerships between organizations that do e-business, adapting links on websites to user click behavior, and deciding on the particular services offered by websites. We implemented a prototype that showed the usefulness and effectiveness of the system. We describe the significance and applications of our approach in e-commerce through information that describes traffic flows between websites.

Keywords: Web server collaboration, Session tracking, Web mining, Clickstream, e-commerce

1 Introduction

With the exponential growth of internet, and the proliferation of web servers, it becomes a must to enhance our understanding of the worldwide network, and to get a global overview of web server interactions and visits. In 2011 the number of live websites reached about 367 Millions and that of registered domains reached 555 Million [1]. In addition, the number of users has also increased significantly since the nineties. This increase shows that studying the different aspects of web surfing, especially Internet flow analysis, is very important for website administrators and managers as it provides insights into the traffic coming to their sites and the traffic leaving it. This will help website providers modify the attributes of their sites, so as to become more effective in widening their customer base, improving their competitiveness, and possibly entering into partnerships with other providers.

Related to our work is Web mining, which is an approach that deals with the extraction of interesting knowledge from the Internet [2]. It has been proposed as a way to collect information from web servers about user interactions over the Internet. Different techniques have been suggested to collect, analyze, and interpret data related to Web servers. Web mining techniques are of three types [3]: web-content mining, structure mining, and usage mining. Content mining extracts information from the storage of the website (text, multimedia, etc.), whereas structure mining works on finding, analyzing, and modeling the structure of relations between webservers. Finally, usage mining studies the interaction between the users and web servers, and it is mainly done through investigating the log files. One of the most important applications of web mining techniques is in E-commerce [4], where the challenges of competing among E-businesses and satisfying users' needs require organizations to devise measures and strategies, which would be most effective if they are based on user behaviors.

Although content and structure mining are important in e-commerce development and management, usage mining remains the basic way of monitoring the evolution of the market, and represents a dynamic reflection of business changes since interactions between businesses and customers cannot necessarily be seen in content and structure mining alone. Usage mining describes inter-page interactions and session streams, and could form a major source of provisions for better services to customers. It could also provide a powerful tool for adapting to market changes, and open the door for e-businesses for analyzing customer behaviors, e-bank transactions, online auctions, blog analysis, and recommendation systems [4].

* Correspondence: hartail@aub.edu.lb
Department of Electrical and Computer Engineering, American University of Beirut, Beirut, Lebanon

The available usage mining techniques are based on clickstream data. A clickstream is a series of page requests triggered by user clicks on webpages, and hence, it represents a record of a user's activity on the Internet. It could include every website and every page that the user visits, how long the user was on a page or site, along with other possible information. Clickstreams can be collected from server log files, but there, human and machine traffic are not usually differentiated. They can also be collected from browsers through Javascript code; which use tracking cookies to generate information sent from browsers to servers. In both approaches, the traffic information is centered around a particular website or a user. Therefore, such clickstream data is limited, and does not totally satisfy the e-commerce requirements since the way such data are available restrict their benefit in applications beyond simple web analysis [5]. The main reason behind the non-utility of the clickstream data remains is the incompleteness of the data, and the huge size of the logged data.

Our work, which we describe in this paper, serves to address the above issues, but more importantly, it describes a collaboration framework to fuse summary information from multiple web servers in order to recognize the paths whose nodes are the webpages that the different user sessions involved. In other words, our method tracks the user's visits to websites through information collected from the servers hosting those sites, without sending Javascript code to web browsers to collect information about clicks. From this aspect, our method is more reliable since Javascript may be disabled by certain browsers. Moreover, there is a fundamental difference between our method and most of the other tracking techniques, in that our goals is to mainly link websites through inflow and outflow traffic from and to other websites, respectively. Our approach preserves users' anonymity since it does not record or need information about users, it is independent of the browser technology, is scalable by virtue of the small-sized data it maintains about a user's trail of visited websites, and is compatible and does not require changes to the HTTP Internet protocol.

The rest of the paper is organized as follows. Section 2 gives an overview of the related literature, while Sections 3 and 4 present the design and implementation of the proposed system, respectively. Next, a load analysis of the cooperating servers and of the Base Server is presented in Section 5, followed by Section 6 that discusses possible threats to validity and their corresponding solutions. Next, in Section 7, a subjective evaluation of the system is described. Finally, the paper is concluded in Section 8.

2 Related work

Since our aim in this work is to investigate interactions among web servers for the end goal of enhancing the functionality of the Internet, and since this is best done by analyzing user interactions with web servers, we restrict our review of previous work to web usage mining. The first step in mining data over the internet is to collect the needed information, which could be achieved through source files on the client, proxy (cache) servers, or web servers [3]. Below is a brief description of each source.

Client log files are based on browser plugins [6], java scripts, and java applets that are integrated with websites. When dealing with code running on the client side, the issue of privacy comes to the surface. Thus, it is recommended that the collection of data streams remains not directly related to users since malicious analysts could benefit from flowing traffic to violate users' privacy [7, 8]. Concerning Proxy log files, they are generated by Proxy servers, which are commonly deployed by organizations to reduce Internet traffic usage. It may appear therefore that Proxy log files should be used along with web server log files to get a better understanding of Internet surfing. In this regard, we will demonstrate that our approach does not require getting hold of data stored on proxy servers, which could be regarded as a desirable feature. Finally, with regard to web server log files, they are automatically generated, and are the most commonly used log files for usage mining. These files do not contain entries of pages served by proxy servers to users, and consequently are not entered into web server log files [3].

Usage web mining undergoes three major steps, namely preprocessing, analysis, and visualization. These are described as follows:

a) Preprocessing: Since web mining techniques deal with log files that contain relevant and non-relevant entries, the first step after the data collection is to filter the data and remove the non-relevant entries [9]. This is also known as cleaning and feature selection.
b) Analysis: After preparing the data, many operations can be applied to benefit from the information it contains. Data that can be deduced include relations among webservers, numbers of visits and users, active or poorly visited links, among others. As was stated in [5], very useful information could be inferred from the interpretation of the clickstreams to enhance the navigation through webpages, and develop approaches to influence the behavior of web visitors. Data analysis is performed by applying statistical techniques, data mining, machine learning, clustering, and pattern recognition to achieve a better understanding of the behavior of users.
c) Visualization: The outcomes of data analysis may not effectively be understood without the visualization of the interactions among webservers, and without the visualization of the path traversals and clickstreams. Visualization could be based on graphs, tables, diagrams, or any mean that allows the derivation of conclusions from the analyzed data [3].

At the heart of our proposed approach is the user session, and one of the first tasks is to define a web session in order to track the user's behavior. A session is most commonly defined as the set of click streams of contiguously visited webpages by a certain user aimed mostly at carrying out a single activity on the Web. It starts when the user opens a webpage residing on some server, and continues to define the stream of clicked links by the same user. Previous research in this area suggested a 5-min inactivity as a mean to end a session [10]. Using this criterion, another work found that over a period of 2 months, a user's click stream could be split into an average of 520 sessions, implying that a typical session would last for about 10 min, and includes about 60 requests to 12 different web servers. Due to the strong dependence on the particular timeout used, the work in [11] tried to find an alternative for defining a session. An algorithm was devised to segment the user's click stream into many logical sessions, and assign Web requests to the session with the most recent use of the referring URL. In this respect, a logical session connects requests related to the same browsing behavior.

Clickstream-based methods of data collection suffer from common weaknesses that make them impractical for use on a large scale basis or for benefiting a wide range of applications. First, some data may be missing, for example due to the fact that some pages are cached on proxy servers [12], and thus causing the web servers that provided the original copies of such pages not to have records of the users' interactions for those pages. This obviously adversely affects the overall visualization and analysis of web clickstreams. Second, recording all requested HTML pages, along with spider requests, could make the logical number of clickstreams saved by a busy web server's log file reach 100 million records per day. Thus, if the mining is taking place on the data collected over 3 months, for example, this would result in a near one Billion records to be fed to the analysis tools. Most of the existing tools would crawl if such a huge data is presented as input [5]. Third, there exists a large number of tools to analyze clickstream data, and those tools could be classified based on the metrics they use to interpret the collected data [5]. The tools may use Web metric-based methodologies [13], basic marketing metric-based methodologies, navigation-based methodologies [14], or traffic-based methodologies [15]. These metrics differ depending on the perspective each tool looks at a webpage, and the targeted types of web services and applications. This implies that there is a lack of a standardized tool to measure the performance of web servers or to have a global view of the interactions over the Internet.

To our knowledge, all existing web usage mining techniques suffer from at least one of the above-mentioned limitations. In this work, we intend to overcome those limitations with a model that services the existing web infrastructure using an efficient and effective mechanism. The importance of our work lies in the fact that web mining is of particular importance to E-commerce, which is one of the major dynamic applications of the Internet. The main two business models of E-commerce are Business to Business (B2B) and Business to Customer (B2C). Web mining could benefit both sectors in a major way, and as such, we will next review the major areas of E-commerce and their applications. We will however discuss the importance of our system for the development and advancement of E-business later in the paper, after describing the design and the operations of our proposed system.

Web based services manage B2C relationships and enhance the sales in E-commerce [16]. Web mining provides insights about customer interactions with web services, and thus helps in developing the e-services capabilities to achieve their goals. More specifically, Web mining provide information that facilitate the study of e-commerce, and the provision of how to develop the web design and services to satisfy customers, and attract more users based on the analysis of user behavior and transactions [4]. E-commerce applications are naturally diverse, and include e-banking, online auctions, online knowledge management, social networking, e-learning, blog analysis, personalization and recommendation systems, and search engines. Web mining could help e-commerce applications in the following way:

a) Web design: improving the Web layout based on path analysis [17].
b) Online auctions: giving an overview that allows the introduction of optimized selling strategies [18].
c) Helpdesks and recommendation systems: reflecting through statistics about which products have better reputation.
d) Business rule generation and recommendations: deducing Web relations and number of visits through analysis of patterns [19].
e) Products browsing: allowing for the automation of shopping and trading by inferring the ranks of different products and services.
f) B2B tasks: improving the search, processing, and coordination among webservers.
g) Knowledge management: learning about users' preferences, thus enabling the management of knowledge to best serve the customers and develop the market [20].
h) Assist products: improving products through analysis of request history over a long period of time [21].
i) Strategic alliances: Providing businesses with data relating to interactions among their web servers, thus enabling them to make educated decisions about merges and collaborations [20].

j) Personalization: exposing the needs and interests of customers, thus enabling e-commerce to meet users' interests and preferences.
k) Advertisement: evaluating the effectiveness of advertisements in promoting commodities [21].

Having established the importance of Web mining for E-commerce and E-services, and having illustrated the limitations of current approaches and tools, we will next describe our proposed approach for efficiently discovering paths of web traffic through web servers, and later, illustrate how the deduced information may be used by businesses to improve their competitiveness and profitability.

3 System design

An overview of the system is shown in Fig. 1. The objective is to construct the sequence of webpages that are visited by the user within a single session, regardless of whether the webpages were delivered by a single web server or multiple ones. To link the set of visited webpages, we used a common identifier, namely the id of the user session, which is a unique identifier generated by the initially visited server. To communicate this identifier across web servers, we use URL Rewriting to inject the unique session id into the hyperlinks in the webpage before sending it to the user. This basically appends the session id to the query string of each of the URLs on the page. An example of an appended id to the query string of a URL is the following hypothetical URL: http://www.fyp.com/reviews.asp?article=24386&sessionid=IE50076359

To collect the records of visited webpages (i.e., clickstreams), we delegated this responsibility to the first visited web server in the session. We made use of the HTTP Referrer header field along with time stamps to construct the sequence of visited pages, and used a message broker to have the second, third, and any subsequent web server send the click information to the first server. The first visited web server stores the records for the particular session in a special database, where they stay there briefly before they get shipped to a central processing server, which we name Base Server.

To temporarily store the clickstream data, each web server (which could be a "first server" for a given session) hosts a database, which comprises two tables: *ClickStream_Data* and *Ship_Data*. The first table contains information about the sessions which started at this server and are in progress (i.e., have not been terminated yet), while the second table contains a list of summaries of completed (i.e., terminated) sessions waiting to be shipped to the Base Server, where they get analyzed and processed along with records from other "first servers". Each record of *Clickstream_Data* comprises the following data about every click on a link in a session (out of the ensemble of clicked links, starting at the server containing this record): the id of the session to which the click belongs (*SID*), a timestamp reflecting the time of the click (*TimeStamp*), the IP address of the server

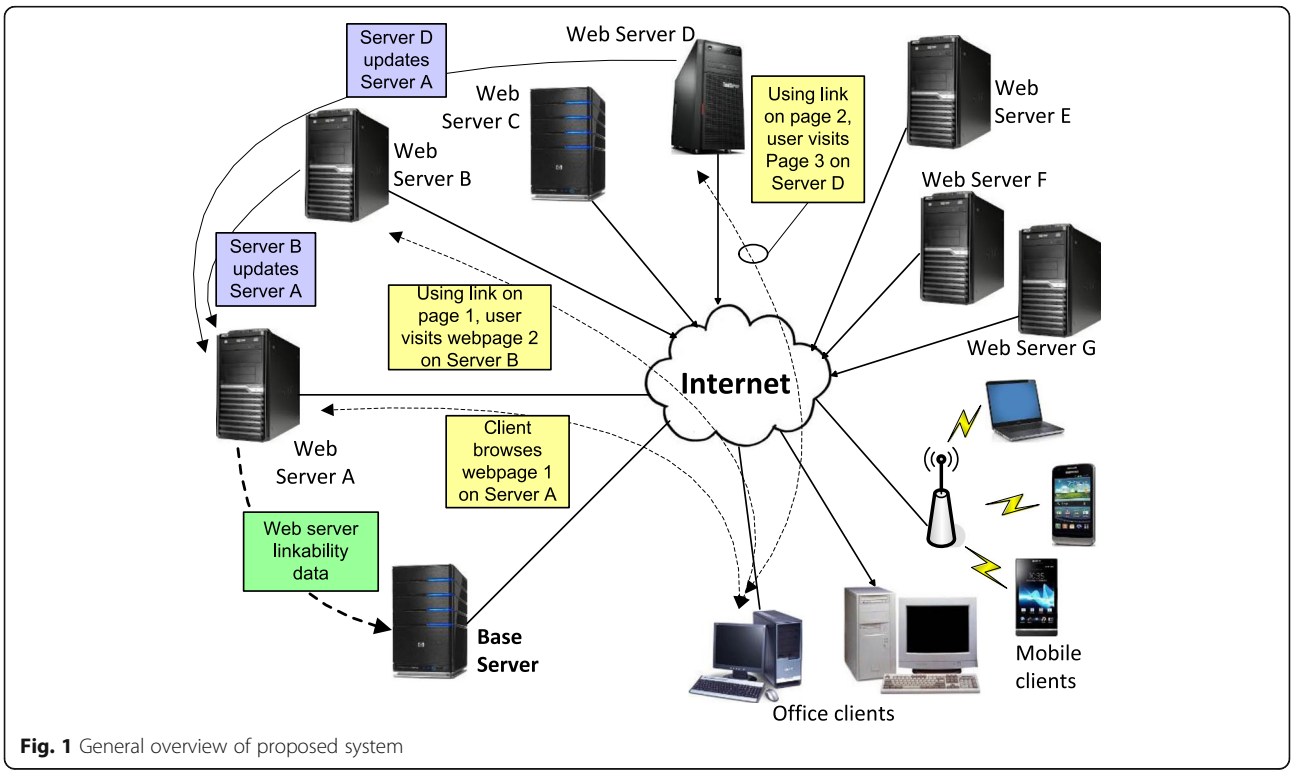

Fig. 1 General overview of proposed system

visited by the user through the click (*ServerIP*), and the address of the referrer (*Referrer*). On the other hand, the second table (*Ship_Data*) contains all the ids of sessions for which the server is the first visited server. Hence in this second table each record will contain the id of the session (*SID*), a flag (*ShipReady*), and a timestamp for the last click in the session (*Tlast*).

The relationship between the two tables is illustrated in the example below (Tables 1 and 2), where two hypothetical sessions are represented.

3.1 System operations

The tracking mechanism is described in the following list of steps in the form of an illustrating scenario, and summarized in the sequence diagram of Fig. 2.

a) The user opens a website residing on server A. His browser issues a GET request.
b) Server A will check if the Referrer field of the GET request contains no URL. Then it retrieves its own IP address - which will be the address of the *first server* in the click stream (because the Referrer is empty) - and appends to it a randomly generated number to form the session id. Server A ensures that this number is unique by consulting the *Ship_Data* table, which contains the ids of all the sessions previously initiated at Server A. The obtained string is used as an id for the session, and is stored in a new record in table *ClickStream_Data* (Session id, server IP, referrer IP address (empty), and time stamp reflecting the time of the start of the session). The server then injects this id into all external links on the webpage, which it sends to the browser. It also sets the Referrer field to its own address.
c) The user clicks on a link on the webpage sent by Server A leading to a website residing on server B.
d) Server B extracts the id mentioned above from the query string, injects it into all external links of its webpage, sets the Referrer field to its own address, and sends back the webpage to the browser. It should be noted that Server B will not add any information to its own *ClickStream_Data* table.
e) In parallel with the above step, or immediately following it, Server B also extracts from the id, the IP of the server initiating the session (i.e., Server A), and uses it to send Server A its own IP address, with a timestamp reflecting the time of the visit to Server B, and Referrer IP address representing the previously visited website.
f) When Server A receives the message from B, it creates in *ClickStream_Data* a new record comprising the IP of server B, the referrer (which is the IP of server A in this case), the received time stamp, and the session id obtained from the query string.
g) After receiving the webpage from Server B, the user may click on a link leading to a website residing on server C.
h) As was the case with Server B, Server C will send the modified webpage to the user after setting the Referrer field to its own address.
i) Server C, which is able to extract the IP of Server A from the query string, sends to Server A its own IP, its referrer IP (which is that of server B in this case), and the timestamp denoting the user's click time.
j) As was the case in Step f, Server A inserts in *ClickStream_Data* a new record comprising the information about the visit to Server C.
k) The process continues until Server A receives no further messages from other web servers for the same session. To do this, it uses a timeout scheme to decide on the end of the click stream session. It is at this point that Server A inserts a record in the *Ship_Data* table, including the information mentioned at the top of this section.
l) Server A, after a short period of time, sends the data to the Base Server, as elaborated in the section that follows.

Table 1 Ship_Data Table

SID	Tlast	ShipReady
8730483841	2016/08/02-18:00:15	0
7465738482	2016/08/02-18:00:40	0

Table 2 ClickStream_Data Table

SID	ServerIP	Referrer	TimeStamp
8730483841	169.254.110.1		10/2/2013 18:00:00
7465738482	169.254.110.1		10/2/2013 18:00:00
8730483841	169.254.47.140	169.254.110.1	10/2/2013 18:00:10
7465738482	169.254.47.140	169.254.110.1	10/2/2013 18:00:40

With regard to the last two steps above, it should be noted that even if Server A receives messages for this session after the mentioned timeout (Step k) or even after shipping the records to the Base Server (Step l), the separate records for the same session will be grouped together and properly ordered at the Base server since the session id is common to all of them, and given the Referrer field and timestamps values.

We mentioned earlier that the Referrer field is used in conjunction with time stamps to properly order the sequence of visited webpages. We demonstrate the necessity of using the Referrer field, since it may seem that timestamps alone can do the trick. For example, considering a situation where a user has opened three additional tabs

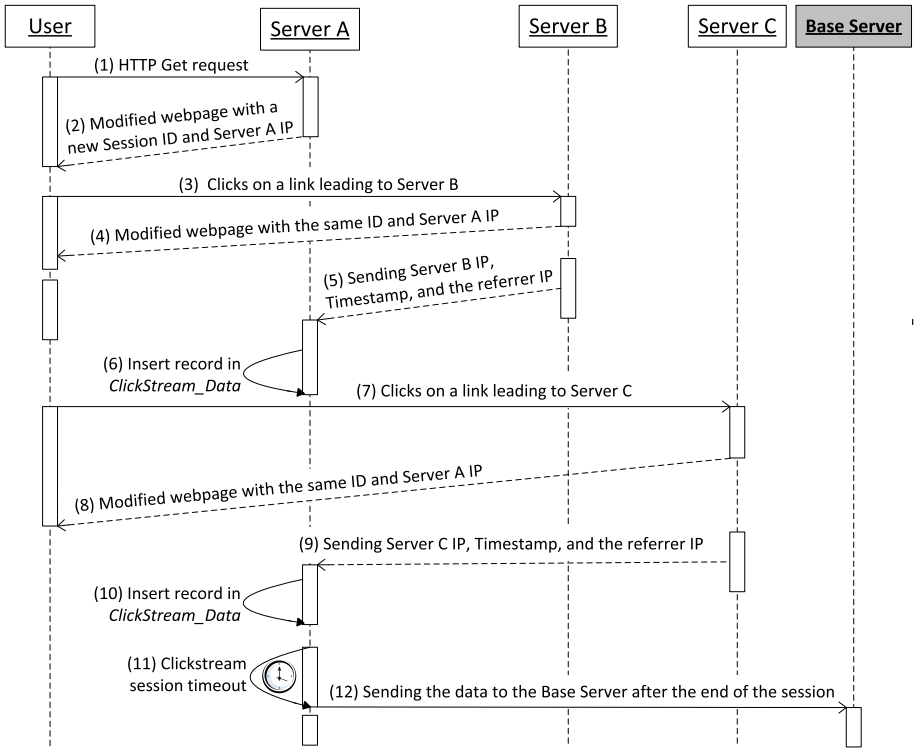

Fig. 2 Sequence of system operations

from links presented on webpage A, without the *Referrer* field, it would be impossible to know that it is the top-right scenario in Fig. 3, which had actually occurred. It is easy to observe that time stamps cannot differentiate between the four scenarios shown in the figure. In the top-left scenario, the referrer to B is A, to C is B, and to D is C. What combines them is the session id, which, along with the referrer value, allows for linking these websites into one clickstream. In the top-right scenario, A is the referrer to all other shown websites, meaning the user has opened three tabs to views their webpages. The bottom left quadrant shows a scenario where B will know that it has been reached through a link on A's page, while both C and D will know that they were reached through B.

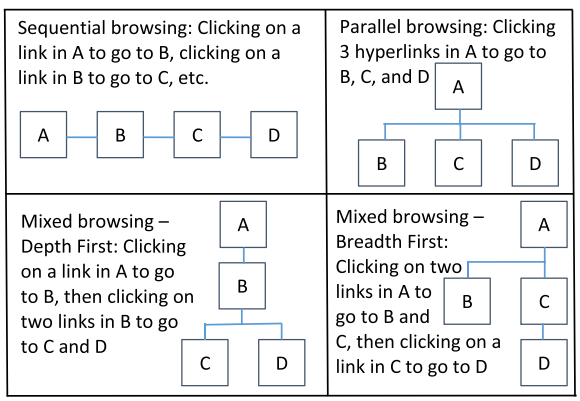

Fig. 3 Probable scenarios for a user visiting four webpages

3.2 Data pooling

We call the process by which the Web servers send the click streams data to the Base Server *the pooling mechanism* (Fig. 4). As explained in the previous section, each server contains a database made up of two tables: *ClickStream_Data* and *Ship_Data*. In addition to the fields which we mentioned previously for the *Ship_Data* table (SID, Tlast), there is also a flag, named S*hipReady*, which is set to 1 for the records in *ClickStream_Data* that have become ready to be shipped to the Base Server. Every time a message corresponding to a click is received by the server (which is the first server in the click stream), a new record in *ClickStream_Data* is created containing the session id, a timestamp reflecting the time of the click, the IP address of the server visited by the click, and the referrer IP address.

For a given session the flag *ShipReady* is initially set to 0. Every amount of time equal to t_{check} (e.g., 5 s), a process on the server (first-server, for the session) will get the timestamp of the latest click in the session, and stores it in the field *Tlast* in *Ship_Data*. After an amount of time equal to $t_{threshold}$ passes beyond the last update of T_{last}, the process sets *ShipReady* to 1. That is, it is when

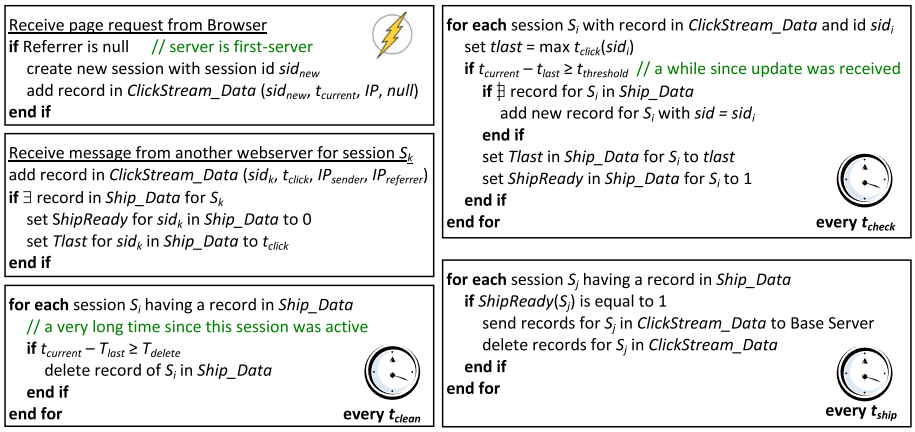

Fig. 4 Pooling algorithm event handlers

$t_{current} - t_{last} \geq t_{threshold}$, where $t_{current}$ is the current time. When *ShipReady* is set to 1, the records of the session in question are ready to be sent to the Base Server at the next time ticket, which will be every T_{ship} (typically every 1 min). But, when an update occurs before sending the data, *ShipReady* will be reset to 0. A process *Ship* that wakes up at every time ticket, checks every time it runs (time ticket) for records whose *ShipReady* is equal to 1. If it finds any, it extracts them from *ClickStream_Data*, and transfers them to the Base Server. Afterwards, the records get deleted from *ClickStream_Data*.

Every time an update to the session is received by the first visited server, T_{last} and *ShipReady* in the *Ship_Data* table will be reset accordingly, i.e., T_{last} will be set to the timestamp corresponding to the last update, and *ShipReady* to 0. In case after the records of a particular session are shipped to the Base Server, a new update is received by the first visited server, a new record will be added to *ClickStream_Data* as usual, and if the record for this session still exists in the *Ship_Data* table (discussed below), the value for *Tlast* will be set to time in the update, and the value of *ShipReady* will be set to 0. Later, after *ShipReady* has been set to 1, the related data in *ClickStream_Data* will be sent to the Base Server at the following time ticket, and added to the previously sent data related to the same session. Hence the clickstream of a session may end up being sent in chunks of records.

If a session witnesses a long period (e.g., 6 h) without any update, the corresponding records will be deleted from *Ship_Data*. As was mentioned earlier, Fig. 4 illustrates the Pooling algorithm. It includes five event handlers that work independently. As implied in the figure, two are event based (top and middle left), while the rest work off timers.

3.3 Operability with proxy servers

In case the user is interacting with the Web through a proxy (cache) server, the proposed design will still work due to the provisions in the HTTP protocol, which were added to help cache servers determine whether the cached webpages they have are stale or up to date. That is, a proxy server upon receiving an HTTP GET command from a browser that is requesting a particular webpage, checks first if it has a cached copy. If it does not have one, or if it has one but its expiration time (Time to Live, or TTL) has passed, it will fetch it from the original webserver (website), and delivers it to the browser after making a copy of it for possible future use. On the other hand, if the proxy server has a non-expired copy, it will check with the webserver to ensure it does not deliver to the user a stale copy. It accomplishes this by sending to the webserver a "conditional GET", which is a GET with an *if-modified-since* header field whose value is the time of when the cached copy was delivered by the webserver. If the server has a fresher copy, it delivers it to the proxy server; else, it sends it a code 304, thus telling it that the cached copy is up to date.

It follows that each time the proxy server receives a request from a user, it contacts the concerned webserver. As the proxy sends the HTTP request to the webserver (GET or conditional GET), it includes in the header the referrer (i.e., the previous webserver) using the Referrer field. Thus when the webserver receives the request, it gets a notification that it has been requested within a session having a given id, and a referrer (i.e., previous) webserver, if any. The following list of steps illustrates how the process works when there is proxy server involved.

a) The user types in a URL, resulting in a GET issued by his browser to the institutional proxy server (PS).
b) If the PS does not have a copy or its copy is expired, it sends a GET request to the Web server (Server A). Else, it sends a GET with the "if-modified-since" field to A.

c) Server A notices an empty Referrer field, so it appends to its address a random number to form the session id (SID). If it has a fresher copy, it modifies the links on the page to include the SID, and sends the page to the PS; else it sends the SID in the 304 HTTP response packet to the PS. In both scenarios, it sets the Referrer field to its own address.
d) If the PS receives a webpage, it stores a copy of this page as is, and delivers it to the browser. Else, it modifies the links on a copy of its cached page to include the SID received from Server A, and sends it to the browser along with the Referrer field value it received from A.
e) The user clicks on a link for Server B, which causes a GET to be sent to PS. The query string is equal to the SID set by A, and the Referrer field is the address of A.
f) If the PS does not have a copy of the webpage at B or its copy is expired, it sends a GET to Server B carrying the SID and the Referrer field value, being the address of A.
g) Server B gets the SID from the query string, extracts from it the first server's address (that of A), and uses it to send A a message whose contents were described earlier. If it received a GET, it injects the received SID into the links of its webpage, sets the Referrer field value in the page to its own IP address, and sends the page to the PS. On the other hand, if it received a conditional GET, it sends to the PS the SID in the 304 HTTP response packet along with the value for the Referrer field (its own address).
h) Upon receiving a response from B, the PS performs the same operations as those when it had received the response from A.
i) The use may click on further links, which leads to similar operations and interactions as those above.

3.4 Base server

The Base Server receives click stream records from the various first-servers, and obviously has to store them in some database for later processing. The database design is usually influenced by the requirements of the data analysis applications and algorithms that are run to generate statistics, inferences, and visualizations. The database may include tables for storing raw data, i.e., data received from the servers, plus computed values that represent summaries and statistics, among others. As was mentioned earlier, the analysis applications that are run on the data may apply statistical techniques, data mining, machine learning, clustering and classification, and pattern recognition functions. The design and implementation of such applications are beyond the scope of this paper, but instead, we give a set of example queries against a sample database that may be executed to generate the desired summaries.

Figure 5 shows the schema of a sample and simple database that could be used on the Base Server to store data and serve the analysis and visualization applications. Below, we list two queries that may be run against this database to generate important inferences.

a) The administrator of a particular website in some organization may be interested to know about the websites visited by users who start at her website, within a particular range of time. Moreover, she wants to know about the frequencies of such visits (i.e., number of times). To accomplish this against the database in Figure 5, we first get the IP address from the Servers Table, and then query the Sessions Table for the session ids, given the address of the server ("First Server") and limiting the visit times ("Start Time") to those falling in the desired time range. After getting the session ids, we query the Clicks table to get the corresponding addresses of visited servers along with the referrers. Finally, these addresses can be used to obtain the URLs information from the Servers Table. In the Implementation Section we illustrate one scenario that visualizes the data returned from such a query.
b) In this second example, the administrator is interested to know about the websites that lead to her website, within a particular range of time. She also wants to know about the frequencies of such visits, as in the previous case. To do this, we first get the IP address from the Servers Table, and then query the Sessions Table for the session ids, given the address of the server ("Last Server"), and limiting the visit times ("Start Time") to those falling in the desired time range. As before, after getting the session ids, we query the Clicks table to get the corresponding addresses of visited servers

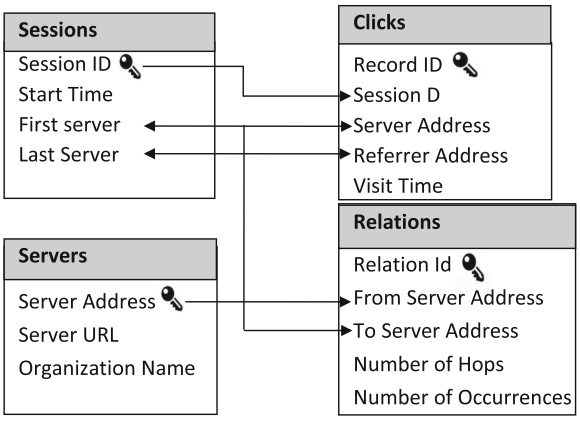

Fig. 5 Simple database used on the base server

along with the referrers. Finally, these addresses can be used to obtain the URLs information from the Servers Table. Similar to the first scenario, we illustrate in the next section one scenario that visualizes the data returned from such a query.

4 Implementation

We implemented a prototype network comprising 7 Web servers, a Base server, and a client by using VMWare to set up the 9 virtual machines. VMware is a software that leverages the power of virtualization to transform datacenters into simplified cloud computing infrastructures and enables IT organizations to deliver flexible and reliable IT services [22]. On each of the seven Web Servers, we have deployed a Web application that contains hyperlinks to all the other servers, including itself.

The applications were developed using Java Server Pages (JSP), where each page contained four java classes: initiation, producer, receiver, and shipment. The first class implements two main functionalities: 1) creating an id for the session at the moment it starts, i.e., at the "first-server", and 2) in the event of a click in the session leading to the server in question, the second functionality is to send the corresponding data to the first-server as elaborated in the design section (session id, IP of visited server, referrer IP, and time stamp).

Hence, the sending is performed in the initiation class, using the producer class, which has the goal of implementing the functionality of Apache ActiveMQ [23], which sends the message to another server, whose functionality is implemented in the receiver class. The last class (shipment) is intended to achieve the pooling of data as was described earlier. It also uses the producer class to send the corresponding data. It should be noted that ActiveMQ is a Message Oriented Middleware (MOM) implementation that is designed for the purpose of sending messages between two distributed applications – in our case, two Web servers. As reported in [23], this MOM is scalable and can handle thousands of messages per second.

As for the "Base Server", it runs a Web Application which includes three java classes, two of which are BaseServer and GraphSet, and implements the database shown in Fig. 5. The BaseServer class runs all the time, and receives data from all possible first-servers. The second class has the function of retrieving data from the database, and creating a list of clicks (Server1 - Server2) which is used to create the final graph, where nodes represent the Web Servers, and the edges' weights represent the number of clicks, which referred the user from Server1 to Server2. Also, this class computes how many times each click has been performed, using the list of clicks generated. The overall network graph is built using NodeXL, which is a free and open-source template for Microsoft Excel 2007, and is used to explore network graphs [24].

In the following we present two scenarios consistent with the ones discussed in Section 3.4 and describe their importance for the development of e-commerce.

4.1 Source servers

A business may elect to evaluate the Internet traffic flow in a network of webservers starting at a particular server. Such data would reveal information about the weight (relative number) of visitors that start at the studied webserver, and gives an idea of how much the other servers are dependent on this server in terms of incoming traffic, directly or indirectly. This is done by simply running the first query that is described in Section 3.4. Once the set of first-servers (SFS) and associated session ids are found, we determine the list of servers that receive direct traffic from them for the found session ids, and keep doing this until we account for all servers that the traffic initiated at the servers in SFS leads to.

The following graph in Fig. 6 shows the outgoing traffic from Web Server D, where the values on the arcs represent the number of distinct sessions. The graph gives several observations. First, Server F seems to be merely a forwarder of traffic of about 25% of Serer D's outgoing traffic, whereas Server G is a sink of most of Server D's traffic. To illustrate the utility of our approach, current tracking methods would identify 50 user sessions leaving Server D to Server G, whereas in reality there are another 151 other sessions that are indirectly reaching Server G from Server D. This would create a greater sense of urgency for the company operating Server D to investigate the reason why such a large percentage of its users (visitors) are going to Server G, where it seems like they are finding the products or services more attractive, or are getting a better deal. The

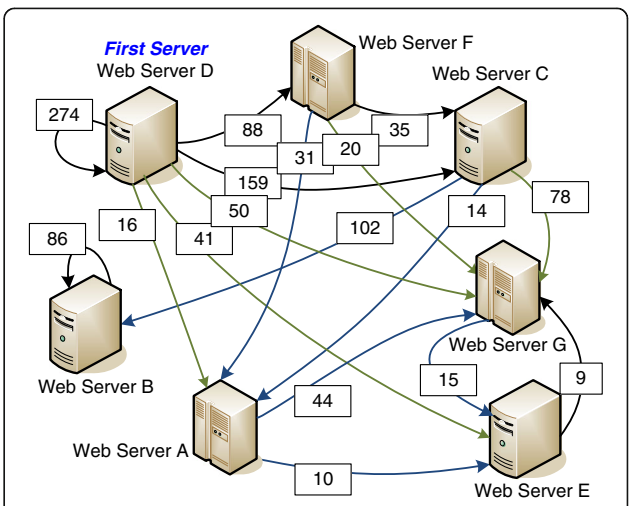

Fig. 6 Traffic originating at Server D

company may decide to take corrective measures, and use our approach to monitor any changes.

Similarly, but perhaps more critically, current systems may not link Server B to Server D, whereas in reality a large number of users are reaching Server B from Server D, and most of them interact with the website and do not end up leaving it.

4.2 Sink servers

A web server may want to know about the servers that lead to it, whether it is an intermediate node or a destination. This can be simply concluded from the base server using the second query in Section 3.4. We illustrate the usefulness of this scenario through an example that is depicted in Fig. 7. By examining the traffic coming to Server A, we observe several facts. First, Servers B and G do not contribute to incoming traffic to Server A, but they forward traffic, especially Servers B. Hence, the links they embed in their webpages and lead to Server A still carry significant importance. Second, Servers D and F are major sources of traffic to Server A, and hence, the operators of Server A should seek ways to attract those users to visit their site directly. This may be done through publishing more related and clearer metadata that search engines can find easier. In parallel, these same operators must ensure that the hyperlinks on sites D and F remain there, or even work on drawing more users from these sites through striking a deal with the administrators of Servers D and F, asking them to add more links or make the existing links more visible.

By generalizing the above two scenarios and the associated visualizations, we can see that the proposed system would enable us to make important conclusions:

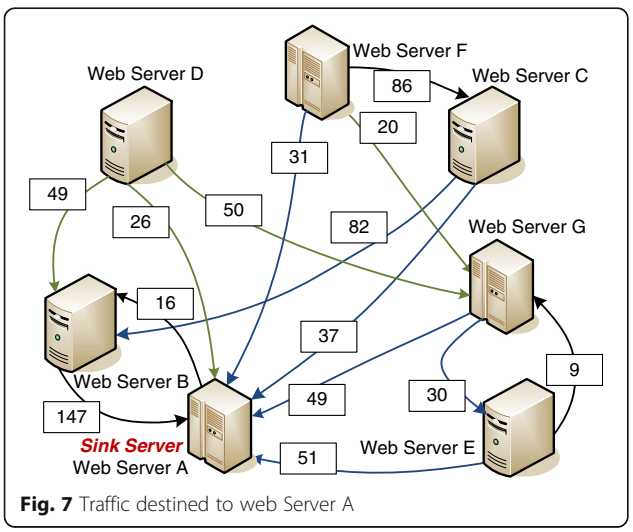

Fig. 7 Traffic destined to web Server A

a) The degree to which a given website (say it is hosted on Server A) is known to users, or how much it is discoverable by search engines, all by looking at the cumulative traffic emanating from Server A.
b) The number and distribution of network traffic flows that end up at other sites give the site administrator important revelations about the types of services and products that may be offered by others, but not offered on their website, or is offered but with disadvantageous characteristics.
c) The above may also be attributed to how the website is designed and how the information is presented and structured. The administrator may consequently decide to change some of the site design parameters in an attempt to retain part of the incoming traffic.
d) The administrator can study in depth the click patterns, and any available information available on those users who stayed at the site and did not navigate away from it through one of its hyperlinks.
e) The Management of the company having the source website may think about entering into a partnership with one or more other companies of the destination websites, if it determines that their services are complementary and that such a partnership is better for the bottom line.
f) After any changes are applied to the website design or content, an updated visualization graph could be obtained to determine if any major differences in the traffic flows have taken place as a result. Hence, our system, with a companion visualization tool, can be used for feedback to illustrate whether the employed strategy by the company in regard to its website is working effectively or not.

A different type of visualization can be a simple listing of statistics and computed values in tables, such as the two ones below. They reveal to the website administrator the other websites that originate traffic to his or her website, and the quantity of such traffic. This can clearly help in devising strategies and action plans for increasing the number of visiting users and expanding the business, as we elaborate further using the data in Tables 3 and 4 (obtained through a different experiment than those that generated Figs. 6 and 7).

The queries that generated the results in Tables 3 and 4 allow the company (whose name is C) to know which webservers are more used to reach its own (e.g., for 192.168.0.2 that would be servers 192.168.0.1, and 192.168.0.3). This could help identifying the design, position, context, and wording of the links that lead to the server. Company C as a result may enter into agreements with other companies owning referring websites with less effective links to modify the design of the links

Table 3 Initial web servers leading to Server B

Web server	Number of sessions
Server B	47
Server A	32
Server C	31
Server F	15
Server D	13
Server E	9
Server G	8

on their webpages so they can potentially forward more traffic, given that these companies are not competitors to C. Moreover, C may be offering some other companies incentives and some form of monetary rewards for including links to its website, but discovers that those links are not essential because the traffic they forward is minimal. This could prompt C to modify the terms of such agreements, or cancel them altogether. Furthermore, if C finds out that the incoming traffic volumes change and are very dynamic, it may elect to go for variable compensation plans that pay per unit of received traffic. This by itself could encourage the companies of the other servers to constantly work on improving their links to become more effective in sending referral traffic to C.

In all, our proposed system can play an important role in increasing revenues of companies by hosting links to other sites that would pay for referred traffic since this brings in more visitors to the site of the paying company. The results (i.e., distribution of traffic flows among websites of companies in the same business) can greatly help in studying the patterns of site visits by users who are seeking a particular service or product. For instance, a given percentage of users makes a purchase or uses an offered service upon the first site visit, whereas another group likes to browse many sites before making the selection. A second application would be to study the effects of local changes (site revisions and upgrades) on the traffic flows, and then take corresponding actions, like moving forward with further changes or reverting back to the old design. Yet another application would be to detect market changes upon introducing a new product or service on the Web based on traffic flow changes. This would enable the organization to react in time by implementing appropriate business changes that align with the traffic changes.

Table 4 Initial web servers leading to Server E

Server IP address	Number of sessions
Server A	65
Server E	56
Server B	44
Server D	32
Server F	18
Server G	15
Server C	14

5 Load analysis

In this section, we study the load on the main elements in our proposed scheme, namely the cooperating servers (First Servers, or downstream servers), and on the Base Server. First though, we list the acronyms used and their descriptions in Table 5.

Table 5 Acronyms used in load analysis

N_{cr}, N_{ps}, N_{fs}	Average number of different visited sites per session, of parallel sessions per client, and of finished unshipped clickstream records
S_{cr}, S_{CST}, S_{SDT}, S_{SCD}	Average size of clickstream record, of clickstream table, shipdata table, and database
S_{SID}, S_{Flag}, S_{time}, S_{IP}	Size of session id, of flag, of time stamp, and of IP address
N_{dv}	Number of daily visitors to a cooperating server
T_{bs}	Length of a browsing session on a server in seconds
N_{CU}	Average number of concurrent users
T_{rs}	Average time it takes for clickstream to be ready to ship to Base Server
μ_P, μ_N	Number of requests per second that processor can serve, and NIC can handle
M_u, M_T	Amount of memory used by server, and total memory
ρ_P, ρ_M, ρ_N	Processor, memory, and network utilizations
λ_P, λ_M, λ_N	Frequency (rate) of requests at the processor, for the memory, and for the NIC
λ_r, λ_a, λ_c	Frequency of record transfers, analysis process executions, and user requests
N_N, N_a, N_c	*Number of concurrent* transfer requests waiting on the NIC, analysis processes execution events, and client requests
B, R_{CS}	Bandwidth, and average bitrate of the Base Server's NIC
T_q, T_d, c_a, c_w	Query formulation, DB query execution, and thread context switch times
T_{tap}	Period of traffic analysis process at the Base Server
ST	Thread stack size
S_q	Combined size of the client query and the resultant database query
H_a, H_c	Memory for click stream records, and for processing client requests
N_l, N_u	Number of records used by analysis process, and transferred from First Servers and not yet processed by Base Server
N_{FS}	Average number of associated First Servers per Base Server
N_{TXfb}	Average number of transfers from a First Server to the Base Server

5.1 Cooperating server load

A cooperating server in our scheme can be either a first server or a downstream server (i.e., the second, third, or further servers visited in a session). If we consider N_{cr} to be the depth of the session (i.e., average number of different visited sites per session), then the probability of a cooperating server to be a First Server within a session will be $1/N_{cr}$, whereas the probability for to be a downstream serer is $(N_{cr}$ -1)$/N_{cr}$.

We denote by the *Servers Collaboration Database* (SCD) the database on the First Server that stores the transient clickstream information for the visiting clients. To compute the average size of the SCD, we first compute the size of the clickstream data per user session. We consider N_{ps} parallel sessions per client, N_{cr} sites per session, and N_{fs} finished sessions whose clickstream records have not yet been shipped to the Base Server. With these definitions, we now can define the average size of the *clickstream* table, and that of the *shipdata* table, respectively as

$$S_{CST} = N_{ps} \times N_{cr} \times S_{cr} \tag{1}$$

$$S_{SDT} = N_{fs} \times \left(S_{SID} + S_{Flag} + S_{time}\right) \tag{2}$$

where $S_{cr} = S_{SID} + S_{time}$ +2× S_{IP}; and S_{SID}, S_{time}, S_{IP}, and S_{Flag} are the sizes of the session id, time stamp, IP address, and ready flag, respectively. Assuming an average of five sites per session and two parallel sessions, the average size of the clickstream table will be $2 \times 5 \times (8 + 8 + 2 \times 16) =$ 480 bytes. On the other hand, assuming an average of two unshipped completed clickstream sessions, the average size of the *shipdata* table is $2 \times (8 + 4 + 8) = 40$ bytes. It follows that the server will have to store about 0.5 KB of click information per user session.

Next, if we denote by N_{dv} the number of daily visitors to a cooperating server, and by T_{bs} the length of a browsing session on a server in seconds (i.e., duration of stay), then the number of concurrent users on such a server can be computed as

$$N_{CU} = (N_{dv}/(24 \times 3600)) \times 2 \times T_{bs} \tag{3}$$

Using the example in the blog of [25], which is based on 150,000 visitors per month and a 10-min stay on the server, the average number of concurrent users (N_{CU}) will only be 69 (will be about 35 if the average stay per site is 5 min). If instead we use an average 100 simultaneous user sessions per server, the memory size requirement will be about 10 Kilobytes per user, computed as follows:

$$S_{SCD} = N_{CU} \times (S_{CST} + S_{SDT})/N_{cr} \tag{4}$$

Next, to compute the traffic (in bits per second) leaving a First Server, we compute the number of transmissions per seconds. With T_{bs} average stay duration on each server, a clickstream record will arrive from the second downstream server T_{bs} seconds after the user has spent T_{bsr} seconds on the First Server, and $2 \times T_{bs}$ seconds later from the third server, and so on. With N_{cr} downstream servers, the clickstream will be ready to ship to the Base Server after $T_{rs} = T_{bs} \times N_{cr}$. Therefore, having N_{dv} daily visits, the number of clickstream table transmissions per second to the Base Server will be N_{dv}/(24×3600), where the size of the table is S_{CST}. In the meanwhile, there are $1/T_{bs}$ single record transmissions per second from the downstream servers.

Hence, the bitrate at a cooperating server's network interface (sent and received traffic), is

$$R_{CS} = (N_{dv}/(24 \times 3600)) \times [(1/N_{cr}) \times S_{CST} + (1\text{–}1/N_{cr}) \times S_{cr}] \tag{5}$$

Using the above expression and the numbers of the earlier scenario (but with a stay length equal to 5 min), the collaborative Server resultant traffic will be 1.12 Kilobits per second (Kbps). By the standard of the example in [25], our scenario corresponds to a very popular site. On the other hand, using a 100 KB page size, and 2 page views per visitor per site, the bandwidth consumption due to *user data* traffic will be 137.5 Kbps. Hence, the additional network traffic (due to our scheme) for such a site will represent an increase of less than 1% in I/O traffic. This highlights the light load that our collaborative servers scheme adds to the Internet backbone. In terms of the cumulative additional traffic in the Internet, the load is only meaningful when it is evaluated through the same communication links, but since the collaborating Web Servers can be distributed throughout the Internet, it will be difficult to compute the traffic on a particular link without understanding the topology of the network through which these servers' traffic flows.

In summary, the above analysis shows that the memory requirements and the additional bandwidth consumption resulting from our proposed scheme is not taxing on the Internet infrastructure, even if the average number of user sessions and other average numbers are larger than the ones we used above.

5.2 Base server scalability

The Base server plays a critical role in our proposed architecture. It serves Web servers, some of which may receive website visits at a high rate, which in turn translates to high volumes of click stream records being shipped to the Base server. To analyze the Base Server's scalability in terms of the number of Web Servers, we abstract the operations of the server main processes, and describe quantitatively the interactions between each process and the underlying hardware resources. In our

analysis, we define the main three hardware resources that affect the server operation: memory, processor, and network. Storage utilization was ignored as it poses no bottleneck in current server implementations. In line with [26], for a smooth server operation and to insure affordable server response time, 1) memory utilization must be below 85% to avoid page faults and swap operations, 2) processor utilization must stay below 75% to make room for kernel and other third party software to operate with no effect on the overall server operation, and 3) network utilization should be kept under 50% to prevent queuing delays at the network interface.

In our analysis, it is convenient to model the processor and network performances using queuing theory, but first, we need to decide on the appropriate queuing model. Considering processor performance, it is well established that an M/G/1-RR (round robin) queuing model would be suitable [27–29]. The M/G/1 queue is a queue model where arrivals are **M**arkovian, the service times have a **G**eneral distribution, and there is a single server. It is designed for round-robin systems (like operating systems) and is generic, as it requires the mean and variance without the full distribution of the service time. This model assumes that requests to the processor follow a Poisson distribution, so that the distribution of the inter-arrival time between requests is exponential with mean λ requests/second, and each request is given a time slice on the processor. Since all requests (click stream record transfers from First Servers, and client queries for traffic flow data) in our case have the same priority, and have low variations in their sizes (i.e., assuming the number of received click stream records from the First Servers does not vary significantly), the queuing model can be reduced to M/G/1-PS (processor sharing). We assume that at full utilization, the processor can serve μ_p requests per second, basically the inverse of a job size, denoted by total processing time. It then follows by queuing theory and little's theorem that the processor utilization, the memory utilization, and network interface card (NIC) utilization are given respectively as:

$$\rho_P = \lambda/\mu_p, \rho_M = M_u/M_T, \rho_N = \lambda/\mu_N \tag{6}$$

where M_u is the amount of memory used by the server, M_T is the total memory, λ is the number of arriving requests, and μ_N is the number of requests that the NIC can be handled. The requests to the network card can be modelled by a Poisson random process, where the service time is constant, basically the transmission delay [30], so an M/D/1 queuing model is appropriate to be applied (the letter "D" denotes **D**eterministic service times). We now derive the utilizations to obtain an expression for the maximum number of simultaneous requests.

a) The click stream records receiving process is multithreaded, where each thread maps to a communication session with a Web server. This process handles λ_r record transfers per second. It does not cause high processing load, but rather a networking load that also affects the memory utilization. The network can serve μ_N communication sessions per seconds, where each session corresponds to an average of N_{cr} clickstream records, each having an average size of S_{cr} KB. Now, assuming the network interface to the Internet has a bit rate of B kbps, we conclude that it can serve $\mu_N = B/(8S_{cr} \times N_{cr})$ users per second. Using the queuing model M/D/1, the number of concurrent transfer requests waiting on the network interface is given by [31]:

$$N_N = \lambda_r \times (2\mu_N - \lambda_r\mu_N)/2\mu_N \times (\mu_N - \lambda_r\mu_N) \tag{7}$$

Each thread will have a stack allocated, and dynamic memory whose size is equal to $S_{cr} \times N_{cr}$ KB. Given that the execution code size is negligible, the memory usage of this process can be approximated as $S_{cr} \times N_{cr} \times N_N$.

b) The statistical analysis process is also multithreaded. It is responsible for keeping the traffic statistics up to date, and may be run according to a time schedule, or triggered by an event (e.g., when the number of received click stream records concerning a given server surpasses a given threshold). This process is assumed to run λ_a times per second. The number of threads is equal to the number of servers having accumulated click stream data, where each thread performs a first stage lookup to associate the click stream data with relevant data in the database, and then runs the analysis on all related click data in the database in order to update the statistics. Each analysis event maps to a thread that consumes memory to maintain its stack, utilizes the processor to run the lookup and the analysis processes, and incurs additional overhead resulting from thread context switches. For the computations, we suppose that the lookup process takes T_l seconds to execute, the analysis process takes T_a seconds, and each thread incurs a context switch of c_a seconds, while thread creation and destruction overheads can be ignored as they are in the order of microseconds [32].

c) The query execution process responds to client (user or application) queries that aim to get a view of traffic flows involving certain Web servers. It is also multithreaded, where the number of threads is the number of clients having pending requests. We assume that the client has a simple API through which he can specify the source or destination Web

server, the flow depth, the time range, or any combination that may also involve other attributes. It follows that it will be the responsibility of the query execution process to formulate the database query out of the received attributes, send it to the database, and then respond to the client with the results. Hence, each connected client maps to a thread that consumes memory to maintain its stack, utilizes the processor to run the query process, and incurs additional overhead resulting from thread context switches. For this process, which occurs λ_c times per second, we suppose that the query formulation takes T_q seconds, while the database execution of the query takes T_d seconds, and each thread incurs a context switch of c_w seconds, as in the above case.

Considering that the second and third processes share the processor, an expression for the number of served concurrent clients N_c and number of concurrent analysis events N_a can be found by applying the expression of the average number of requests in the processor [33]:

$$N_a = \lambda_a / \left[\lambda_a \mu_p / (\lambda_a + \lambda_c) - \lambda_a{}^2 \mu_p / (\lambda_a + \lambda_c)\right] \quad (8)$$

$$N_c = \lambda_c / \left[\lambda_c \mu_p / (\lambda_a + \lambda_c) - \lambda_c{}^2 \mu_p / (\lambda_a + \lambda_c)\right] \quad (9)$$

It follows that the total memory used by the second and third processes above is

$$M_u = N_a \times H_a + N_c \times H_c + (N_a + N_c) \times ST \quad (10)$$

where ST is the thread stack size; H_a is the dynamic memory allocated for the click stream records (the records that must be loaded into memory from the database for lookup, and for the records necessary for analysis to update the traffic flow graph statistics), and H_c is the memory allocated for the received client request plus the database query. With the average record size and the average number of accumulated records at a First Server being S_{cr} and N_{cr} (defined earlier), and the number of records used by the lookup process (N_l), then the value of H_a can be approximated by $(N_N \times N_{cr} + N_l) \times S_{cr}$. On the other hand, the value of H_c is simply the combined size of the client query and the resultant database query denoted as S_q.

We now use the above definitions to develop expressions that lead to a measure of the load on the server. To start with, the CPU can serve μ_p record processing events per second:

$$\mu_p = 1/\left(T_l + T_a + T_q + T_d + c_w + c_a\right) \quad (11)$$

From before, the processor utilization is ρ_P, which must be less than 0.75, or else, the processor will be the bottleneck and will limit the server's scalability. Here we rewrite ρ_P in (6), after setting λ to $\lambda_a + \lambda_c$ as:

$$\rho_P = (\lambda_a + \lambda_c)/\mu_p \quad (12)$$

Next, the total memory usage of the server processes (specified in (10) earlier) is

$$M_u = N_a \times (N_N \times N_{cr} + N_l) \times S_{cr} + N_c \times S_q + (N_a + N_c) \times ST \quad (13)$$

The memory utilization of the processes is then given by $\rho_M = M_u/M_T$ and must be below 0.85.

Finally, the utilization on the external network interface is given by

$$\rho_N = (8S_{cr} \times N_{cr})/B = 1/\mu_N \quad (14)$$

$\rho_N = (8S_{cr} \times N_{cr})/B$ and should be below 0.5. The expressions of ρ_P and ρ_N are linear in λ ($\lambda_a + \lambda_c$ and λ_r, respectiely), and their solutions yield $\lambda_P < 0.75/(T_l + T_a + T_q + T_d + c_w + c_a)$ and $\lambda_N < B/(8S_{cr} \times N_{cr})$, respectively. On the other hand, the expression of ρ_M is cubic in λ, and its solution λ_M, if it exists, is in the form $\lambda_M < \lambda_{M1}$ and $\lambda_{M2} < \lambda_M < \lambda_{M3}$, or $\lambda_{M1} < \lambda_M < \lambda_{M2}$ and $\lambda_M > \lambda_{M3}$, where λ_{M1}, λ_{M2}, and λ_{M3} are the possible solutions of $\rho_M - 0.85 = 0$. The solutions are shown in the left chart of Fig. 8 as plots of the utilizations versus λ (λ_r, λ_a, λ_c) by using values for the other parameters in accordance with the literature. To explain these results, we consider the same values for the record size (S_{cr}) and number of records (N_{cr}) in a session (48 bytes and 5, respectively) as before, and having an available transfer rate of 10 Mbps. For processor utilization calculations, we assumed the cumulative value of $T_l + T_a + T_q + T_d + c_w + c_a$ to be 100 milliseconds, while for memory utilization, we considered $N_l = 500$ records, $N_N = 10$, in addition to the value of 5 for N_{cr}.

We are now ready to plot the hardware utilization results using the derivations above:

- The processor utilization, ρ_P is specified in (11) versus λ and related to μ_p in (10).
- The Memory utilization is given in (6) as M_u/M_T, and is related to λ via N_a in (8) and N_c in (9), which are in turn used in the expression of M_u in (13).
- The Network utilization is given in (14) as $1/\mu_N$, and is related to λ_r via N_N in (7), where the latter is a function of both μ_N and λ_r.

The left part of Fig. 8, which shows the hardware limitation results, indicates that the processor could form the bottleneck of the Base Server at high job rates, where a job could be the processing newly received clickstream records from a First Server, running the traffic analysis process when $N_N \times N_{cr}$ reaches a certain threshold, or answering a user query that is asking for

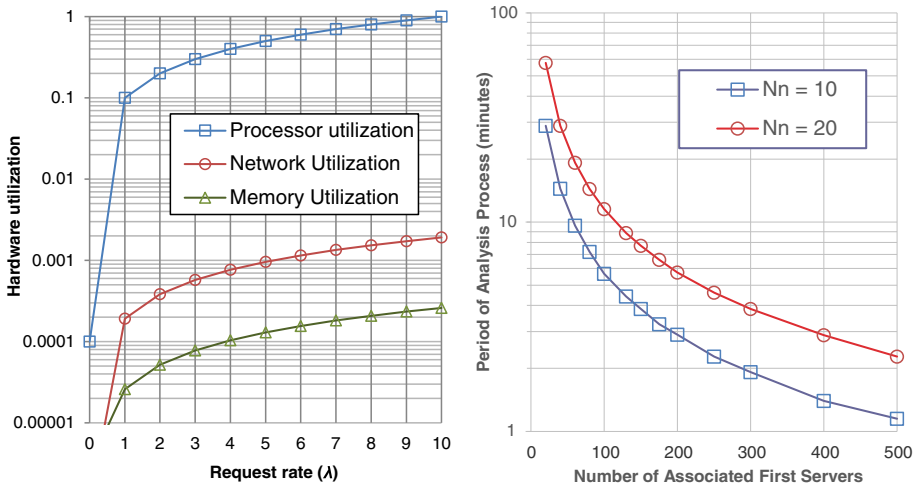

Fig. 8 Hardware utilization (*left*) and probability of simultaneous access (*right*)

some traffic results. Out of the above three processes, the second one is the most compute bound and is therefore the most taxing on the server's processor since it involves updating the traffic results. The graph illustrates that the processor can handle up to 8 concurrent jobs (value of λ when the processor utilization reaches 75% of capacity). Hence, given the average number of visitors at each web server, we study next the largest period of the traffic analysis process ($T_{tap\text{-}MAX}$) at the Base Server so as to avoid queuing at the processor. We do this for different numbers of associated First Servers per Base Server (N_{FS}), and considering two values for N_N: 10 and 20 (thus resulting in 10×5 and 20×5 records to be processed, respectiely). The average number of transfers from a First Server to the Base Server is $N_{TXfb} = (1/N_{cr}) \times N_{dv}/(24 \times 3600)$. Hence,

$$T_{tapMAX} = \max\left(T_{tap}\right) \left| \left(N_{FS} \times N_{TXfb} \times T_{tap}\right)/(N_N \times N_{cr}) \leq 8 \right. \quad (15)$$

Considering the earlier scenario with 5000 daily visits per web site (N_{dv}), we compute T_{tapMAX} for different values of N_{FS} and N_N. The results are displayed in the right graph of Fig. 8. The importance of these results lies in the fact that they describe the length of the period for updating the traffic analysis results without overloading the processor. The graph also illustrates that a single core processor can handle 500 web servers comfortably (with a period that is upward of 1 min) even by running the process after a small set of records (e.g., 50 records) is received from associated web servers (First Servers). We should additionally note that a multicore processor, with x cores, is expected to be able to handle x times the capacity above.

6 Threats to alidity

In this section we discuss the factors that could reduce the effectiveness of our proposed scheme or threaten the validity of its operations and output. We identify two such factors: 1) the existence of non-cooperative web servers that take part in the users' clickstreams, and 2) the necessity to have multiple Base Servers to handle the load from the First Servers, thus giving rise to an issue dealing with the inclusion of same servers in different clickstreams distributed across different Base Servers. We describe how the first issue is resolved by virtue of the operations of the system, and how the second issue can also be resolved through a proposed mechanism.

6.1 Non-cooperative servers

Our solution depends on the notion of cooperation among web servers in the Internet, in that servers will need to be setup to send and receive from each other user clickstream information, and also send to the Base Server completed clickstram records. As we elaborate in the next section, there are added benefits that websites will get from this solution, and hence they can choose whether to "join in" by implementing the needed changes, or not. The system as a whole (network of cooperating web servers) can still operate if not all the browsed servers within a session support the functionality. In this regard, there will be two possible scenarios in a session. In the first scenario, an initial subset of servers from the set of visited ones in a session do not support the cooperation functionality in our scheme. In this case, if the first visited server does not support the functionality, the second visited server will be considered the First Server, if it supports the functionality. Similarly, if the second server does not support it, then the third server may be the First Server, and so on. In the second

scenario, there is a gap in the set of visited servers that support the cooperation functionality. That is, a supporting server is followed by one or more non-supporting servers. Although both scenarios will result in missing clickstream information, and do not convey the complete picture about the visited websites, but they will not break the system.

From a technical standpoint, a First Server is marked by the absence of the Referrer field in the GET request, and this will enable our system to keep working through the first scenario. Similarly, the end of the clickstream is marked by the timeout attribute of our scheme, and hence it will protect against the possibility of visited servers within the same session not sending clickstream information to the First Server. Note that even after a timeout, and after the First Server sends the clickstream records to the Base Server, if a particular Web Server sends its clickstream info to the First Server, this information will still be forwarded to the Base Server and joined to the records that were already shipped. This is explained toward the end of Section 3.1.

6.2 Multiple base servers

According to Fig. 8, the processor forms the bottleneck at the Base Server, after the number of concurrent combined jobs (updates from First Servers, statistical analysis tasks, and user requests for summaries) reaches 80, thus causing the processor to hit the 75% utilization mark. To sole this scalability problem, a simple solution would be to design a queue at the Base server in which all jobs get queued before they are served, thus mitigating the possibility of overrunning jobs. This solution however can lead to large delays that are associated with many jobs waiting in the queue when the load is high at the Base Server.

A better and obvious solution is to install multiple Base Servers to serve the collaborative web servers throughout the Internet, and distribute them geographically to sere nearby servers. This association of web servers to Base Servers is a clustering problem in which the clusterheads are the Base Servers. Although clustering solves the capacity issue, it gives rise to an issue, dealing with handling inter-cluster browsing traffic. That is, there will be user sessions that will involve cooperating servers belonging to different clusters. In such scenarios, the First Servers of those sessions will report their respective clickstream records to their corresponding Base Servers. The challenge becomes what to do for user queries about browsing traffic originating or terminating at particular servers, when related traffic records are distributed over multiple Base Servers. To overcome this challenge, we propose for each server when reporting a clickstream record to the First Server to also include along with its id the id of the Base Server it belongs to its cluster. With this, when the First Server sends the ids of the involved servers in the session to the Base Server, the latter can identify those servers that are not in its cluster, and can therefore associate them in its database with their respective Base Servers. With this information available locally, when it runs its traffic analysis process, the Base Server can query the concerned Base Servers for clickstream traffic records that are related to servers involved in its own unprocessed records.

We explain this scheme with the help of Fig. 9, which shows a scenario of two clickstream records generated at different times; the bottom one first, and then the top one (as implied by the step numbers). The top one is sent by Server A (playing the role of a First Server for this clickstream) to Base Server 1 (BS1; the Base Server it is clustered with), and the bottom one was sent by Server E (the First Server for this clickstream) to Base Server 2 (BS2).

In the figure, the association of each server with a Base Server is indicated in parentheses. Upon receiving a clickstream record (Step 1 for BS2, and Step 4 for BS1), the Base Server informs the other Base Servers that are associated with servers in this record that it holds traffic information about servers in their clusters. In the example of the figure, in Step 2, BS2 informs BS1 that it holds traffic data about servers A and B; and tells BS3

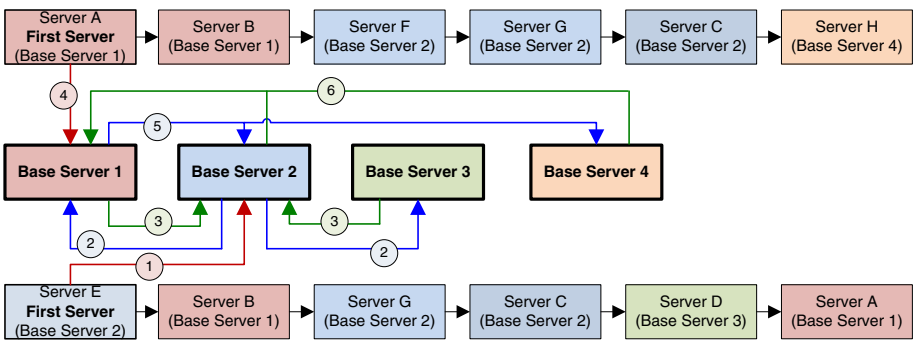

Fig. 9 Servers in records on multiple base servers

that it has data about server D. At a later time, after BS1 receives the top record in the figure from the First Server (Server A) in Step 4, it informs BS2 that it holds traffic data about servers F, G, and C; and tells BS4 that it has data about server H (Step 5). Assuming that prior to the top session taking place, only the bottom session occurred (thus resulting in the bottom shown clickstream being transferred from Server E to BS2), then subsequent to BS1 updating BS2 and BS4, BS2 will reply by sending to BS1 (Step 6) the bottom shown record since it includes Servers A and B. This will hence allow BS1 to include the bottom record in its analysis when it runs it. On the other hand, BS4 will reply with an empty list.

7 Subjective system evaluation

As was illustrated, this proposed framework could introduce major impacts as an Internet Flow Analysis system. It would allow website providers to achieve major goals. The advantages that our approach offer over previously proposed web mining techniques can be summarized by the following:

7.1 Accuracy

In previous web mining techniques, path completion was used as a way to compensate the lack of some log entries (missing is due to the fact that proxy servers may find the data in their cache, thus preventing webservers from recording the visit entries). The work in [3] mentioned that using the referrer field and the topology of the website heuristics can be used to suggest missing entries in order to be able to build the graph of webservers network. In our model, we solve the issue associated with proxy servers by virtue of our approach and design (refer to Section 3.1).

7.2 Scalability

Our model does not affect the scalability of the internet network. Simply, if a webserver is introduced to the map, it will send its entries to the Base Server, just like any other web servers does. The scalability of the Base Server was studied in Sections 5 and 6, and additionally, it is worth pointing out that in contrast to other techniques that mine massive log file data, the base server in our approach only has to process summary data stored in a database of very few tables, where indexing techniques could be employed to speed up the access to the data, and the execution of the queries.

7.3 Compatibility and integration

The proposed system design can smoothly fit into the existing infrastructure of the Internet, since it does not require any modifications to the HTML protocol, nor does it add any content to the webpages, other than the minor "control" information, i.e., appended session ids to the hyperlinks, and the added referrer field values in the HTTP header.

7.4 Privacy

Some web mining techniques use plugins on the user side, where such plugins are considered trusted. Beside the concern over malicious plugins that may violate users' privacy, supposedly trusted plugins could still uncover user identities, and send them to the party responsible of mining the collected data. Moreover, even when using the web server log files rather than client side plugins, log files contain sensitive information about IP addresses of users and browser histories. Such information can be used to profile users and uncover them [7, 8]. On the other hand, our proposed system preserves the anonymity of users since the data sent to the Base Server are only related to the session id and visited webservers, and does not include further private information that could be used to trace the identities of users.

7.5 Cost and time effectiveness

Since cost and time are tightly related, current techniques necessitate the preprocessing of collected huge data sets before analyzing the relevant remaining entries. Such preprocessing takes time and sometimes requires manual intervention in order to remove some irrelevant entries, and to complete the missing paths. Our system does not require such preprocessing, simply because it is not starting from huge data. The Base Server only receives relevant records and can work on them directly or as a background process. Moreover, several mining techniques that use server log files were limited to the collection of few-days of data in order to minimize the probability of privacy disclosure, and in order to handle the massive amount of data. Furthermore, previous server log file mining techniques first collect data from many major agents before applying the pre-processing on the whole data. Such a multi-step process costs time and money since some web servers do not give free access to their log files.

7.6 Dynamicity

The way the base server gets the collected information allows for both offline and online processing. That is, it can process the data while receiving new entries, and may choose to delay the integration of the new data into the results (e.g., graphs).

7.7 Cleaning issue

As was discussed earlier, current web usage mining mechanisms depend on the pre-processing of data before performing the analysis on the log files. It follows that the filtration of the huge amount of collected data was essential in order to remove the non-necessary information. In our design, some of the cleaning procedures are no more needed since the model does not introduce this problem, while others could be solved the same way they were solved before:

a) Multimedia and script files: some scripts or image files are downloaded without the requests of users. The related sites are usually listed in the log files and need to be removed after detection based on file extension [3]. Such a problem is not present in our case since the webserver only sends to the Base Server the clickstreams information related to the users' requests.
b) Non-available pages: such entries could be present in log files and need to be removed based on the HTTP status code (which is set to the corresponding error number) [3]. In our design, simply if the resources are not available, the webserver would not send the corresponding entries to the Base Server.
c) Crawlers and Spiders: due to the wide expansion of the Internet, programs have been built up to automatically search the Web. Spiders, crawlers, webbots, web agents, web robots, and others are being used for such a mission [34, 35]. In the log files, in order to differentiate between a user request and an "automatic" click, one could inspect the host name and the agent field since they usually declare themselves as agents. Thus, by string matching in the agent field the related entries, automatic clicks can be detected [3]. In our design, this problem can be solved at the webserver side by the same mean, which enables it to know it should not send the entry to the base server.

7.8 Improving E-advertising

An e-business that finds out that its website is heavily being accessed through another website could as a result advertise on this other website. For example, if the administrator of website A gets to know that it is greatly referred by website B, she can study the background of this website and advertise on similar websites.

7.9 Offloading common services to cloud servers

Web servers that have services in common (inferred from the referral traffic) can partner to establish or outsource to cloud servers that provide common services or products that are identified to cause the referral traffic. This can help improve scalability and reduce traffic in the Internet through offloading common services to the Cloud, and potentially share the cost. Moreover, organizations in such a situation could share the cost of renting out a cloud server to host their services.

7.10 Improving competitiveness

This may be realized through checking services provided by the "visited-through" (one or several-hops-away websites), for the purpose of comparing them with own services to improve them.

7.11 Improving search engines

This may be realized by including in the search results partner websites. For example, if a search engine displays in some results website A, it may also display most visited websites that are referred by A.

7.12 Imposing fees on referrals

An e-Business can make profit based on the referral volumes to other e-Business websites. It can also relate the most frequent visits to offered services and increase their fees, or can strike partnerships with other websites if it turns out that they form a major source of traffic into its website.

8 Conclusion

Our proposed model overcomes the limitations of previous usage web mining techniques on a multidimensional scale. We demonstrated the model's importance for e-Commerce both from an economical point of view and from a technical aspect. This system, if thoroughly integrated into the Internet, it could create new opportunities for expanding the range of services, improving competitiveness, increasing revenue, and fostering partnerships that allows web sites to complement or supplements each other's services. On the social side, the study of traffic flows on the Internet would have a huge impact. By understanding the source and type of incoming traffic, organizations' officials and web site administrators can understand their customers better, and can cater their marketing strategies accordingly. Nevertheless, many other benefits can be realized through the various types of statistics that can be derived from the data at the Base Server.

Abbreviations

API: Application programming interface; M/D/1: Markovian arrivals, Deterministic service times, and 1 server; M/G/1-PS: Markovian arrivals, general distribution service times, 1 server - Processor Sharing; M/G/1-RR: Markovian arrivals, General distribution service times, 1 server - Round Robin; MOM: Message oriented middleware; NIC: Network interface card; PS: Proxy server; SFS: Set of first-servers; SID: Session identifier; TTL: Time to live

Acknowledgement

This work was supported by a generous grant from the Lebanese National Council for Scientific Research (LNCSR) under Grant G/3526 - 25/11/2015. The grant money was used mainly to pay for stipends (student employment) to the three students.

Authors' contributions

Hassan Artail is the main contributor of this work, given that he originated the idea, provided the general design, and wrote most of the paper. Ammar El Halabi, Ali Hachem, and Louay Al-Akhrass all worked on this as a capstone project, and equally contributed to the implementation and testing of the developed system. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Received: 1 September 2016 Accepted: 15 December 2016
Published online: 15 January 2017

References

1. Brain S. Total number of websites. 2012. Available: http://www.statisticbrain.com/total-number-of-websites/.
2. Etzioni O. The World-Wide Web: quagmire or gold mine? Commun ACM. 1996;39:65–8.
3. Varnagar C, Madhak N, Kodinariya T, Rathod J. Web usage mining: a review on process, methods and techniques. In: Proceedings of the 2013 International conference on information communication and embedded systems (ICICES). 2013. p. 40–6.
4. Ting I. Web-mining applications in e-commerce and e-services. Online Inf Rev. 2008;32:129–32.
5. Sen A, Dacin P, Pattichis C. Current trends in web data analysis. Commun ACM. 2006;49:85–91.
6. Song H, Chu H-H, Kurakake S. Browser session preservation and migration. In: Poster Session of WWW. 2002. p. 7–11.
7. Chen T, Han W-L, Wang H-D, Zhou Y-X, Xu B, Zang B-Y. Content recommendation system based on private dynamic user profile. In: Proceedings of the 2007 International conference on machine learning and cybernetics. 2007. p. 2112–8.
8. Choi J, Lee G. New techniques for data preprocessing based on usage logs for efficient Web user profiling at client side. In: Proceedings of the 2009 IEEE/WIC/ACM International joint conference on Web intelligence and intelligent agent technology, vol. 3. 2009. p. 54–7.
9. Srivastava J, Cooley R, Deshpande M, Tan P-N. Web usage mining: discovery and applications of usage patterns from web data. ACM Sigkdd Explorations Newsletter. 2000;1:12–23.
10. Qiu F, Liu Z, Cho J. Analysis of user Web traffic with a focus on search activities. In: WebDB. 2005. p. 103–8.
11. Meiss M, Duncan J, Gonçalves B, Ramasco J, Menczer F. What's in a session: tracking individual behavior on the web. In: Proceedings of the 20th ACM conference on Hypertext and hypermedia. 2009. p. 173–82.
12. Goldberg J. Why web usage statistics are (worse than) meaningless. Available online, retrieved October 2013 from http://www.goldmark.org/netrants/webstats/.
13. Clifton B. Advanced Web metrics with Google analytics. 3rd ed. Mississauga: Wiley; 2012.
14. Fu Y, Sandhu K, Shih M. A generalization-based approach to clustering of web usage sessions. In: Web usage analysis and user profiling. Springer Berlin Heidelberg; 2000. p. 21–38.
15. Moe W, Fader PS. Capturing evolving visit behavior in clickstream data. J Interact Mark. 2004;18:5–19.
16. Singh M. E-services and their role in B2C e-commerce. Manag Serv Qual. 2002;12:434–46.
17. Yu H, Huang X, Hu X, Wan C. Knowledge management in E-commerce: a data mining perspective. In: Proceedings of the International Conference on Management of e-Commerce and e-Government (ICMECG'09). 2009. p. 152–5.
18. Tu Y. An application of web-based data mining: selling strategies for online auctions. Online Inf Rev. 2008;32:147–62.
19. Yang H, Wang C-S. Locating online loan applicants for an insurance company. Online Inf Rev. 2008;32:221–35.
20. Wenyun L, Lingyun B. Application of Web mining in E-commerce enterprises knowledge management. In: Proceedings of the International Conference on E-Business and E-Government (ICEE). 2010. p. 1769–72.
21. Mei L, Cheng F. Overview of Web mining technology and its application in E-commerce. In: Proceedings of the International conference on computer engineering and technology (ICCET), 2010. 2010. p. 277–80.
22. Lowe S. Introducing VMware vSphere 4. 2009.
23. Snyder B, Bosnanac D, Davies R. ActiveMQ in action. Manning, Shelter Island, NY; 2011.
24. N. Graphs. The social media research foundation. http://nodexl.codeplex.com/.
25. WhoIsHostingThis.com. How much bandwidth does your website really need? http://www.whoishostingthis.com/blog/2010/04/14/bandwidth-needed/.
26. Citrix, Consulting. MetaFrame XP oracle 11i application scalability analysis. 2003. http://www.dell.com/downloads/global/solutions/MetaFrame_XP_Oracle_11i_Application_Scalabilit_Analysis.pdf.
27. Cao J, Andersson M, Nyberg C, Kihl M. Web server performance modeling using an m/g/1/k* ps queue. In: 10th Int'l Conf. in Telecommunications, 2003. ICT; 2003.
28. Gupta V. Finding the optimal quantum size: sensitivity analysis of the M/G/1 round-robin queue. ACM SIGMETRICS Perform Eval Rev. 2008;36:104–6.
29. Willig A. A short introduction to queuing theory. 1999. http://www.telecom.otago.ac.nz/tele302/ref/Willig_ch1n2.pdf.
30. Kurose J, Ross K. Computer networks and the internet. In: Computer networking: A Top-down approach. 7th ed. London: Pearson; 2016.
31. Dattatreya G. Performance analysis of queuing and computer networks. Boca Raton: Chapman & Hall/Crc Computer & Information Science Series; 2008.
32. Ling Y, Mullen T, Lin X. Analysis of optimal thread pool size. ACM SIGOPS Oper Syst Rev. 2000;34(2):42–55.
33. Kleinrock L. Time-shared systems: a theoretical treatment. Journal of the ACM (JACM). 1967;14:242–61.
34. Birukou A, Blanzieri E, Giorgini P. Implicit: a multi-agent recommendation system for web search. Auton Agent Multi-Agent Syst. 2012;24(1):141–74.
35. Chau M, Chen H. Personalized and focused Web spiders. In: Web intelligence. 2003. p. 197.