

# Real-Time Monitoring and Alerting of Web Usage

Andrew Morrison  
Department of Computer Science, RPI  
Troy, NY 12180

John Punin  
Oracle Corporation  
Redwood Shores, CA.

Mukkai S. Krishnamoorthy  
Department of Computer Science, RPI  
Troy, NY 12180

## Abstract

*Web usage mining is an active research area for its uses in web site maintenance and for the potential economical impact. In the past, research has focused on off-line statistical analysis, learning the user behavior and on identifying most frequently visited structures. Our paper takes an ambitious approach by proposing and studying effects with an on-line monitoring mechanism of web usage. In our system named  $W_{live}^3$ , we devise efficient real-time algorithms for identifying most visited sites and site-paths in real time. We also include features such as an advance warning when there is a potential denial of service attack. The  $W_{live}^3$  system uses LOGML [3] and the graph library of the WWWPal suite [2], and lay the foundation for future developments.*

*We conclude by analyzing performance properties of our system and propose some suggestions for future work.*

Studies and research in web usage mining have addressed these scenarios in the past. Traditionally this has been accomplished by gathering data on off-line statistics, user sessions, most visited web pages, most traversed web paths etc [3], [4]. Our paper addresses a more ambitious goal: that of identifying and classifying the visited web sites, traversed web paths in real-time. At the Computer Science Department, Rensselaer Polytechnic Institute, we have built a tool  $W_{live}^3$  that addresses these issues.

$W_{live}^3$  has been developed an open source software implemented in C with both a console and GNOME interface. While monitoring real-time events of web site usage it also provides advice as well as an instant warning mechanism to web managers. These features are extremely useful especially in preventing denial of service attacks by knowing the load on web pages (explained later in the paper). Our system also lays the foundation for future developments to enhance the existing framework.

## 1 Introduction

Popularity of a web site is in most part related to the content, presentation and construct of a web site. For commercial web sites this is of prime importance as each visit indicates a potential consumer. The consumer might either buy, be tempted to buy or recommend the site to others. This factor results in a potential for increased traffic to a web site which in turn can result in advertising revenue.

In case of commercial web sites, web managers work with designers, content producers, and programmers to popularize their web sites. Web site statistics such as URL location, number of hits determine site usage as well as web site traffic. While web managers prefer to have large number of users to visit their pages, there are times when users orchestrate a flood of visits to prevent others to visit a specific web site. This is damaging as this not only paralyzes the web site but also could result in a loss of revenue stream.

## 2 Previous Work

Web usage mining is the process of studying and discovering web user behavior from web log data. Usually the web log data collection is done over a long period of time (one day, one month, one year, etc). Later, three steps, namely, preprocessing, pattern discovery and pattern analysis [5] are carried out. Preprocessing is the technique of transforming the raw data into a usable data model. The pattern discovery step uses several data mining algorithms to extract the user patterns. Finally, pattern analysis reveals useful and interesting user patterns and trends. These steps are normally executed after the web log data is collected.

$W_{live}^3$  extracts and analyses user patterns in real-time, as the web log data is collected. Several commercial applications, such as Kantara [6], Deepmetrix [7], ClickTracks [8] and CacheFlow [9], have implemented real-time features to report user sessions. Kantara monitors and tracks

the users as they access the web pages. Visitor Intelligence from DeepMetrix uses real-time processing to update the business metrics to identify business trends. The web site can be modified to improve those business metrics. Click-Tracks is installed in web pages to see user patterns. Finally CacheFlow Reporter generates User, Network Traffic, Security and Top Ten summaries Reports from web log data. CacheFlow has also real-time monitoring features. Interesting work has also been done for distributed real-time web usage mining [11] for frequent behavior patterns.  $W_{live}^3$  not only reports user patterns using LOGML (standard XML application), it also visualizes user patterns (user sessions web graph).  $W_{live}^3$  is designed for research purposes and it is an open source application.

### 3 Realization of the System

$W_{live}^3$  is realized as a set of open source tools. Its core components are as follows.

- **w3ld**: The  $W_{live}^3$  daemon accepts extended log file entries as they are generated and builds the structures describing the current state of the web server.
- **w3lc**: The  $W_{live}^3$  client is a console program for communicating with w3ld.
- **w3live**: w3live is a GNOME graphical interface providing easier access to the core features of  $W_{live}^3$  as well as providing visualization of web user graphs.

Figure 1 describes the system architecture.  $W_{live}^3$  reads web server logs in the extended log file format [1].

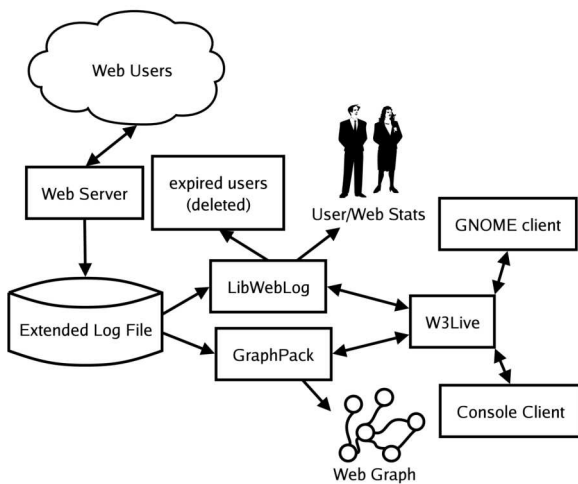


Figure 1.  $W_{live}^3$  Architecture

Building on previous work, GraphPack [10] and WWW-Pal are employed for the collection and analysis of log data. Figure 2 shows the realized  $W_{live}^3$  user interface.

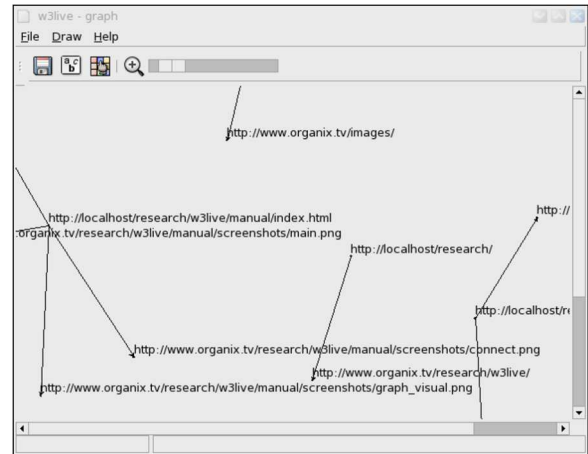


Figure 2. Graph of user clicks through website

#### 3.1 Example Analysis

The  $W_{live}^3$  software attempts to give insight into how web users are using a web site and the relationships between site structure and success in finding information.

The web site of the department of computer science at Rensselaer Polytechnic Institute serves as an example for studying real-time web usage. The site <http://www.cs.rpi.edu> serves approximately 100 requests per minute during daytime activity. It primarily serves as space for computer science faculty and students to serve class information, research and personal information.

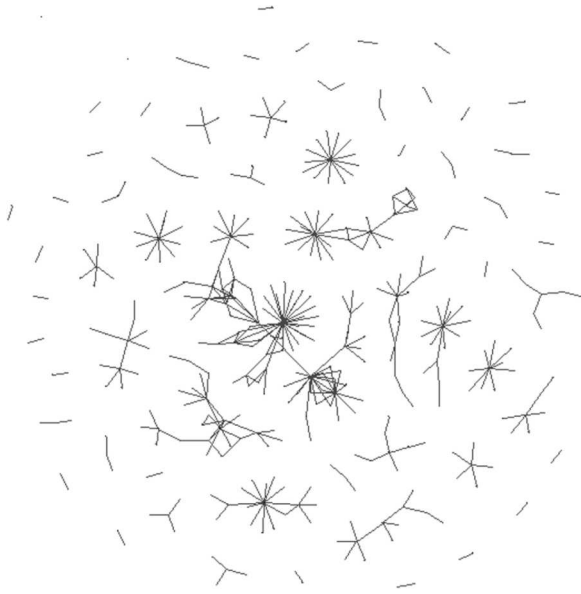


Figure 3. User Search Path

Through  $W_{live}^3$  web user graph visualization we are able to note various search patterns and judge the effectiveness of the site structure.

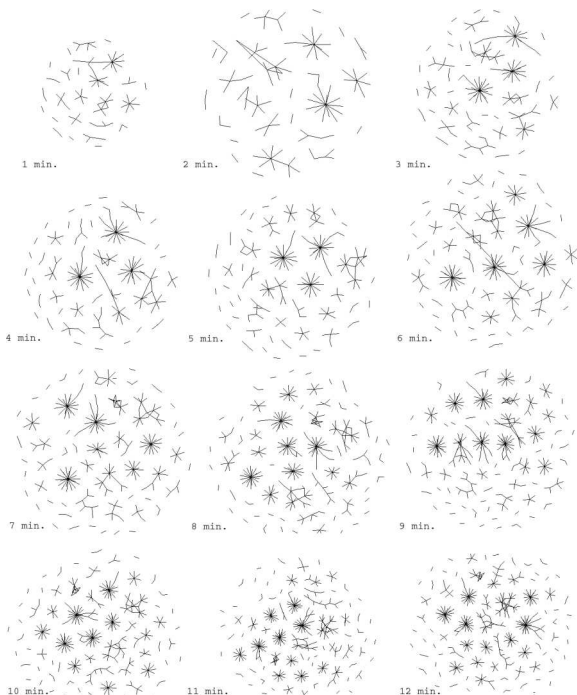
Figure 3 illustrates a user search path generated using  $W_{live}^3$ . A popular search engine has imprecisely referred the user to the site and has left the user to perform further search. A Web Manager presented with such information will be able to better mold web site structure to the user's search styles, or design better search engines based on how users are looking for information.

Activity over a 10 minute period is shown in Figure 4.



**Figure 4. Snapshot of Site Activity**

Immediate insight into the nature of user activity on the web site is given. The ad-hoc structure of the department web site is immediately clear. A Web Manager may use such information in determining the success or failure of the global web site structure.



**Figure 5. Evolution of user paths**

Figure 5 shows the evolution of user paths over time (from top left to bottom right). In this analysis of [www.cs.rpi.edu](http://www.cs.rpi.edu) over five minutes, we see several small hubs emerging and the lack of site-wide coherence becoming obvious.

## 4 Algorithms Used

The following algorithms are realized as  $W_{live}^3$ , a set of programs for real-time web usage analysis. Figure 6 gives an overview of the data acquisition loop, whereby log entries are processed into more meaningful structures. The functions **Process Log Entries** and **Terminate Expired Users** are expected to run concurrently.

Each iteration begins by updating a graph of nodes representing requested document and edges representing the relationship between a referring and target document. Building on previous work we employ the GraphPack library [2] for storage and manipulation of graphical data.

Following the graph update we update site wide statistics and user statistics. We employ WWWPal's WebLog library for this task. It is important to note that the WebLog library does much more than is needed for our task, and thus we get a slight performance degradation in these superfluous computations. Section 5 further describes such considerations.

In the following sections we describe algorithms for discovering unusual or perhaps harmful site activity.

### 4.1 Explosive Popularity

Whereby a denial of service attack may be confused with legitimate usage, it is worthwhile to differentiate the good from the bad traffic. A sudden explosion of popularity is likely due to an unexpected link from a site with more traffic. We would therefore like to determine if a single document is receiving an abnormal amount of traffic. Sudden explosive popularity may overload a web server and cause it to stop responding. Early detection would allow some time for a Web Manager to reroute requests to another server, or allow for a scaled down version of a document to be served.

We assume document popularity is governed by a Zipf distribution [12]. Each document requested is assigned a value representing its load.

Let  $d$  be the current document and  $t$  be the request count. The function  $ltime(d)$  indicates the last time document  $d$  was requested. The load  $\ell$  of document  $d$  at iteration  $t$  is computed as follows

$$\ell_t(d) = 1 + \frac{\ell_{t-1}(d)}{(now - ltime(d)) + 1} \quad (1)$$

In order to see if  $\ell_t(d)$  represents normal traffic we would like to compare it against a Zipf (power law) distribution of load averages site wide. To do this we measure the distance

```

Process Log Entries
while(entry = Read Next Log Entry)
  entry data = extract log entry data(entry)
  Lock Data(mutex)

  update web graph(entry data)
  {
    referrer = find node(entry data.referrer)
    if(referrer not in graph)
      referrer = create node(
        entry data.referrerURL)

    target = find node(entry data.target)
    if(target not in graph)
      target = create node(entry data.target)

    O(1) {
      edge = find edge(referrer,target)
      if(edge not found)
        edge = create edge(referrer,target)

      update node(target,entry data)
      update load average(target)

      update edge(edge,entry data)
      update graph load average(
        target.loadAverage)
    }

    O(1) {
      update user stats(entry data)
      {
        user = find user(entry data.address)
        if(user not found)
          user = new user(entry data)

        O(1) {
          update user stats(user,entry data)
          update user load average(user,
            entry data)
          update site load average(
            user.loadAverage)
        }
      }

      Unlock Data(mutex)

      test for events()
      {
        O(1) {
          if((graph.loadAvg -
            threshold · graph.Std Dev) -
            target.loadAvg < 0)
            ALERT!(user is hogging bandwidth)

          if((site.request rate -
            threshold · site.Std Dev) -
            request rate < 0)
            ALERT!(current target is receiving
            abnormally heavily load)
        }
      }
    }
  }

Terminate Expired Users
while(1)
  O(|users|) {
    wait USER TIMEOUT seconds
    Lock Data(mutex)
    for each user in the list of users
      O(1) {
        if(last access time(user) >
          USER TIMEOUT)
          terminate(user)
      }
    Unlock Data(mutex)
  }

```

**Figure 6. Statistics Updating Loop with Time Complexity Overview**

from the current load average,  $\ell(d)$ , to the greatest load average,  $\ell(d_{\max})$ , and compare it to an average load average, represented by the logarithmic difference between the max load average  $\ell(d_{\max})$  and the min load average  $\ell(d_{\min})$ .

$$\log \frac{\ell(d)}{\ell(d_{\max})} > \log \frac{\ell(d_{\max})}{\ell(d_{\min})} \times \frac{1}{\gamma} \quad (2)$$

Given a threshold  $\gamma$ , we may say  $\ell_t(d)$  represents explosive popularity if  $\ell_t(d)$  is more than  $\frac{1}{\gamma}$  times the slope of the power law curve further from the largest ‘normal usage’ load average

In the next section we find that explosive popularity and denial of service attacks may be confused. One should be given explicit precedence over the other so that a triggered alarm will be mutually exclusive among the two.

## 4.2 Denial of Service Monitoring

An average denial of service attack on a web server will be detected by a sudden overpowering flood of requests. Commonly these requests come from an array of machines running synchronized programs whose job is simply to make requests.

We would like to examine the temporal spacing between requests. We assume a Zipf distribution and look for spots when the space between requests is abnormally small.

Let  $c_t$  represent the time of a request at iteration  $t$ . The minimum length of time between perceived requests depends on the maximum between how quickly a web server can generate log file entries, and how quickly a single entry can be processed. The representation of  $c_t$  should reflect this minimum time in accuracy. It is likely that seconds will be too large, and milliseconds will be sufficient.

In order to tell if an unusual request rate is occurring we may look at perturbation in the log-log slope of occurrences of times between requests. It will be sufficient to count only the occurrences of the minimum and maximum of times between requests.

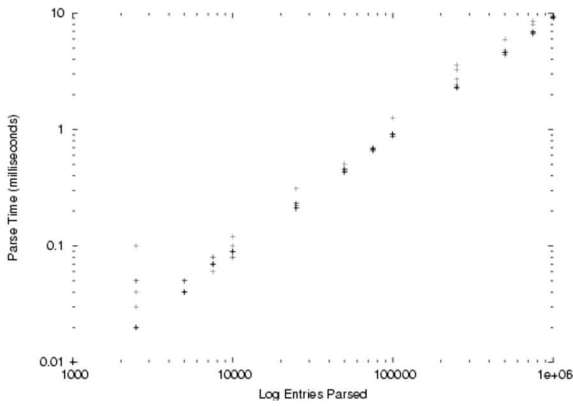
For  $\delta_t = c_t - c_{t-1}$ , let  $\delta_{\min}$  be the minimum  $\delta_t$ , and  $\delta_{\max}$  be the maximum  $\delta_t$ . Given  $\mathcal{O}(\delta_t)$ , the occurrence count for  $\delta_t$  we look for instances where

$$\log \frac{\mathcal{O}(\delta_{\max})}{\mathcal{O}(\delta_{\min})} < \log \frac{\delta_{\max}}{\delta_{\min}} \times \frac{1}{\gamma} \quad (3)$$

In situations where Equation 3 are true for a given  $\gamma$ , and Equation 2 are false, it may be said that a denial of service attack is in progress, and appropriate action may be taken.

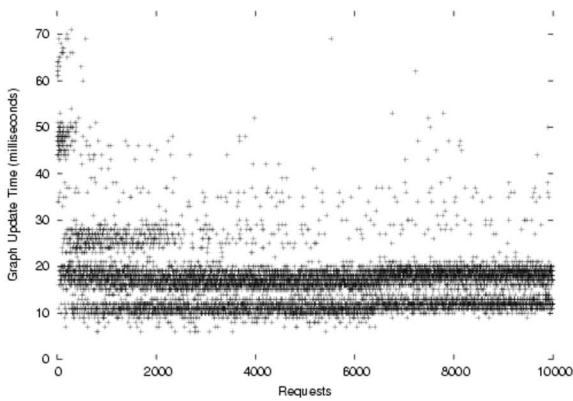
## 5 Performance of the System

The  $W_{\text{live}}^3$  software is tested for performance and comparison against the stated algorithms. For ease of implementation the stated algorithms are not followed explicitly. We show divergence between possible time complexity and real testing and explain how the system may be tuned.



**Figure 7. Log Parsing given as time to complete  $n$  log lines on a logarithmic scale.**

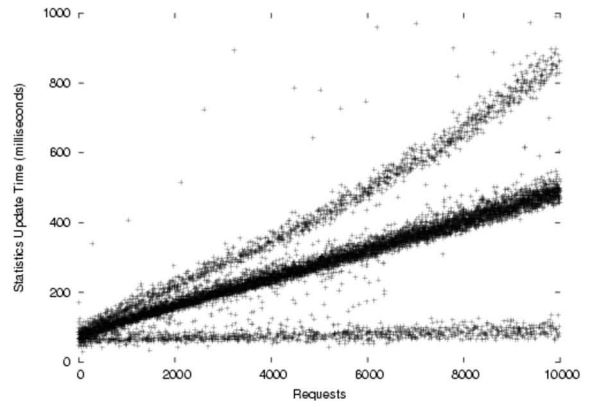
As log entries are read, we must first parse the data back out of string format. Figure 7 shows the time required to compute  $n$  entries on a logarithmic scale. Log processing on a whole has linear complexity while the processing of a single entry is  $O(1)$ .



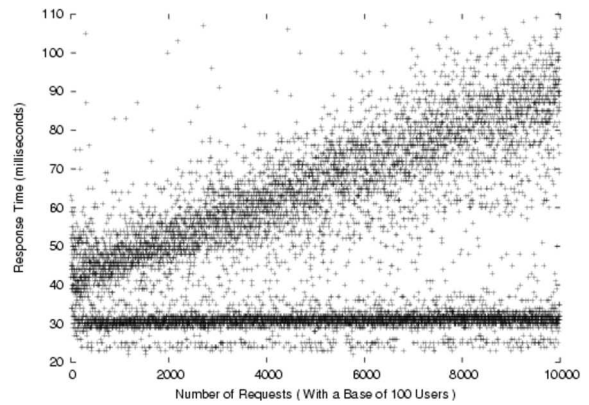
**Figure 8. Time in milliseconds for web graph update for each request**

Figure 8 shows the time in milliseconds to update the web graph at each iteration. The stratification in Figure 8 is attributed to the varying cases whereby any combination of adding a new target node, adding a new referrer node and adding a new edge may occur. Further, the initial clustering of data points between 40 and 50 milliseconds is attributed to the necessity of each request resulting in the addition of new nodes and edges.

Figure 9 shows the time in milliseconds to update the site and user statistics. It is important to note that more



**Figure 9. Statistic Update Times With Constantly Increasing Number of Users**



**Figure 10. Statistic Update Times With a Constant 100 Users**

and more users and documents are being added as more and more requests are being made and thus the complexities being represented are a function of request count, user list count and document list count.

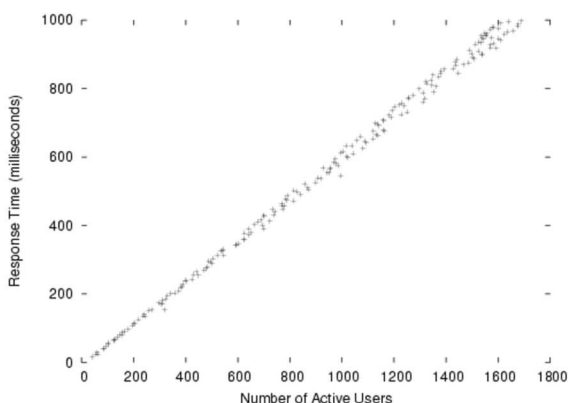
The linear complexity represents the vast majority of requests. The linearity is produced by scans over the list of users, both as searches and scanning for users who are likely to have finished their sessions.

The  $O(1)$  complexity represents file transfer errors, user requests not of type GET or unrecoverable log entry anomalies. In each of these cases, minimal statistical updating is performed.

The higher stratum represents a worst case of  $O(users^2)$  complexity occurring when a user's session is terminated. The list of users is scanned on occasion for user's whose sessions have expired. If a session is found

to be stale, they are removed from a list, requiring another scan through the list of users. This is the price paid for building on top of an off-line mining package and falls into the class of needed improvements.

Figure 10 represents the time complexity for updating statistics with a constant base of 100 users. In this case the linear time complexity is attributed to the growing number of documents being tracked as the web graph. The relatively slow growth is not noticeable in Figure 9 as it is dominated by the increasing difficulty of user management as more and more users are added.



**Figure 11. Times for Building the List of Active Users**

When studying web usage, perhaps the most common task is to request a list of currently active users. Figure 11 shows the number of active users on a server versus the number of milliseconds needed to compile the list of users and respond. We see that compiling the list is  $O(|users|)$  as expected for such a task.

## 6 Conclusion

This paper presents a system for efficient real-time monitoring of web sites. The advantages of this system are:

- It is an open source system.
- It is easily extendible to include monitoring of new events.
- It provides a real-time visualization of user behavior.
- It provides advance warnings of denial of service attacks.

Future research will focus on further improving the efficiency of weblog library. We are also planning on enhancing  $W_{live}^3$  system to capture the semantics of the visited structures in real-time. Semantic information such as

common keywords, geographic location and external referers will help to offer more accurate advance warnings and improve visualization. It will also lay the foundation for future semantic web mining research.

## References

- [1] Phillip M. Hallam-Baker, Brian Behlendorf. *Extended Log File Format W3C Working Draft WD-logfile-960323* <http://www.w3.org/TR/WD-logfile.html>
- [2] Punin, J. R., *WWWPal Suite for Analysis and Organization of Web Sites*, Ph. D. Thesis, August 2003, RPI, Troy, NY.
- [3] Punin, J.R., M. Krishnamoorthy and M. Zaki, LOGML - Log Markup Language for Web Usage Mining, in *WebKDD Workshop of SIGKDD*, 2001.
- [4] Cooley, R., B. Mobasher and J. Srivastava, Web Mining: Information and Pattern Discovery of the World Wide Web, in *Proceedings of the 9th IEEE International Conference on Tools with AI*, November 1997.
- [5] Srivastava, J., R. Cooley, M. Deshpande, P-T. Tan. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data, in *Proceedings of SIGKDD Explorations*, 2000.
- [6] Kantara <http://www.kantara.co.uk/>
- [7] DeepMetrix Mining - Visitor Intelligence Service <http://www.deepmetrix.com/>
- [8] ClickTracks - <http://www.clicktracks.com/>
- [9] CacheFlow Reporter - <http://www.cacheflow.com/>
- [10] Krishnamoorthy, M.S., F. Oxaal, U. Dogrusoz, D. Pape A. Robayo, R. Koyanagi, Y. Hsu, D. Hollinger and A. Hashmi. GraphPack: Design and Features. *Software Visualization*, vol. 7, pages 83–100, 1996.
- [11] Florent Masseglia, Maguelonne Teisseire, Pascal Poncellet. Real Time Web Usage Mining: A Heuristic Based Distributed Miner, in *Proceedings of the RIDE 2002*, 2002.
- [12] Steven Glassman, A caching relay for the World Wide Web, *Computer Networks and ISDN Systems*, vol. 27, pages 165–173, 1994.