

WebMon: A Performance Profiler for Web Transactions

Thomas Gschwind*
tom@infosys.tuwien.ac.at

Kave Eshghi
kave@hpl.hp.com

Pankaj K Garg
garg@hpl.hp.com

Klaus Wurster
klaus_wurster@hp.com

Hewlett Packard Labs
1501 Page Mill Road
Palo Alto, CA-94304

Abstract

We describe WebMon, a tool for correlated, transaction-oriented performance monitoring of web services. Data collected with WebMon can be analyzed from a variety of perspectives: business, client, transaction, or systems. Maintainers of web services can use such analysis to better understand and manage the performance of their services. Moreover, WebMon's data will enable the construction of more accurate performance prediction models for web services.

Current web logging techniques create a log file per server, making it difficult to correlate data from log files with respect to a given transaction. Additionally, data about the quality of service perceived by the client is missing entirely. WebMon overcomes these limitations by providing heterogeneous instrumentation sensors and HTTP cookie-based correlators. In this paper, we present the design and implementation of WebMon and our experience in applying WebMon to an HP Library web service.

1 Introduction

WebMon is a performance monitoring tool for World Wide Web transactions. Typically, a web transaction starts at a Web browser and flows through several *components*: the Internet, a web server, an application server, and finally a database. Web site administrators currently have no tool to determine transaction residence times in each of these major components. WebMon overcomes this limitation.

Monitoring web transactions poses several challenges:

Instrumentation The web is a heterogeneous system: components can range from a variety of browsers (Netscape's Navigator, Microsoft's Internet Explorer,

Opera), web servers (Apache, IIS, Netscape), and application servers (HP's Application Server, BEA's WebLogic, BroadVision). Hence, one challenge of monitoring web transactions is to develop *transparent* instrumentation to monitor a transaction as it is flowing through these *heterogeneous* components.

Correlation The web is characterized by the mixture of heterogeneous components that are dynamically brought together to service a transaction. Client browsers are usually not under the control of the same organization that is running the web server or the application server. Moreover, a single web request may include results of multiple web servers all under the control of different organizations. To provide meaningful performance profiles, however, the data obtained from these multiple, diverse components has to be correlated.

Performance and Scalability As with any profiling tool, an important challenge is to construct the sensors and collecting agent as unobtrusive to the main operations as far as possible. Hence, profiling and collection should not impose an undue overhead on the real system.

WebMon addresses these challenges in the following ways: it provides a variety of instrumentation techniques to monitor typical components participating in a web transaction. We pass special correlators with HTTP cookies to perform data correlation. Since all web transaction components allow cookies to flow through them, and forward the cookies to the next component, this correlation approach works across a variety of components. For performance and scalability, we use an extensible sensor-collector architecture [10]. Each sensor is a simple probe that monitors a transaction as it is flowing through it, reads and possibly modifies appropriate cookies, and sends an asynchronous message to a collector. For scalability, WebMon can be configured for multiple collectors or a single collector. The

* Visiting researcher from Technische Universität Wien, Austria

collectors may or may not be located on the same machines as the sensors.

Previous approaches to monitoring of web transactions either take a systems approach, i.e., monitoring the CPU, disk, memory, and network utilization on the servers, or monitoring of individual components of the servers [11]. From previous work we know that systems' monitoring is not enough to enable accurate performance models for distributed systems [7]. Additionally, component data by itself is hard to correlate. WebMon embodies a correlated-transactional view of the performance. A WebMon sensor monitors the response time from the start of a transaction, i.e., a user's browser, and correlates such measurements with residence time data from other sensors at the web and application servers. This data can be used by software performance modeling techniques, e.g., the method of layers [14] or layered queueing models [16], to develop more accurate predictive models for such systems.

WebMon uses an *event-driven* rather than *sampling* style of performance monitoring. Hence, the sensors at various points of a web transaction are triggered by the transaction flowing through them. In this manner, WebMon resembles the approaches to performance monitoring enabled by the Application Response Monitoring (ARM) proposed standard [8]. This approach, however, is in contrast to tools such as PerfMon [1] that monitor based on a sampling interval. Event-driven approaches suffer from the limitation that as the volume of events increases so does the monitoring overhead. For sampling approaches one can decrease the sampling frequency as the transaction volume increases. In future we will implement dynamic throttling of data collection to address this issue [5].

The rest of this paper is organized as follows. Section 2 explains the basic components of WebMon and our scheme for measurement correlation. Sections 3 demonstrates an example use of WebMon. In section 4 we describe some related work. Finally, we conclude in section 5 with our main results and directions for future work.

2 The WebMon Architecture

WebMon relies on a sensor-collector architecture, as shown in Figure 1. Sensors are typically shared libraries or script components that intercept the actual processing of a request. Each sensor is logically composed of two sub-sensors: the *start* sensor and the *end* sensor. The start sensor implements the correlation aspect of the monitoring, and the end sensor forwards monitored data to a collector that gathers and correlates the data. Since web browser are frequently behind a firewall, we use a surrogate web server to forward the browser's data to a collector. This web server can be the same as the original web server serving the web pages, or a different one. The collector makes the data avail-

able in a file, database, or a measurement server, for further processing.

Transactional analysis requires the ability to uniquely identify each request, enabling the correlation of the data collected by each server. Hence, the start sensor of the browser generates a request identifier and passes the identifier along with the request to the web server. The end sensor sends the performance data to the surrogate web server. Web browsers are instrumented indirectly by instrumented the HTML pages returned by the web service.

The start sensor of the web server extracts the identifier from the request, and passes the identifier along to the application server. The web server's end sensor sends its performance along with the request identifier to the WebMon collector. Similarly, the application server WebMon sensor processes the request and send its own data to the WebMon collector. The collector correlates the data received by the individual components on the basis of the unique identifier associated with each request. In our prototype implementation we use a single collector, although multiple servers tailored for different deployment architectures are feasible.

To explain the basic correlation mechanism, we define the following terms:

Web Transaction A request by a user, that starts at a browser (by the user 'clicking' on a URI, or typing in a URI in a browser window, or selecting a bookmarked page), goes through the Internet (or Intranet), sent to the web server, and possibly forwarded to multiple back-end servers.

Requested Page The HTML page that is the object of the given web transaction.

Referrer Page The HTML page, if any, on which the user issued a "click" to obtain the requested page.

Given the nature of real-world demands for low overheads on web transactions, instrumentation for web transaction monitoring may not be in place all the times. Hence, the referrer page may be instrumented or not, the requested page may be instrumented or not, and the web (and other back-end) servers may be instrumented or not. Our correlation scheme has to take into account each of these situations and a combination thereof.

We will first explain the scenario in which all the components are instrumented, and then describe situations in which one or more components are not instrumented. Also, we will first describe the situation with only the web browser and server, and subsequently extrapolate to application servers.

At the heart of the correlation scheme we use two cookies to transmit information among the various components of a web transaction:

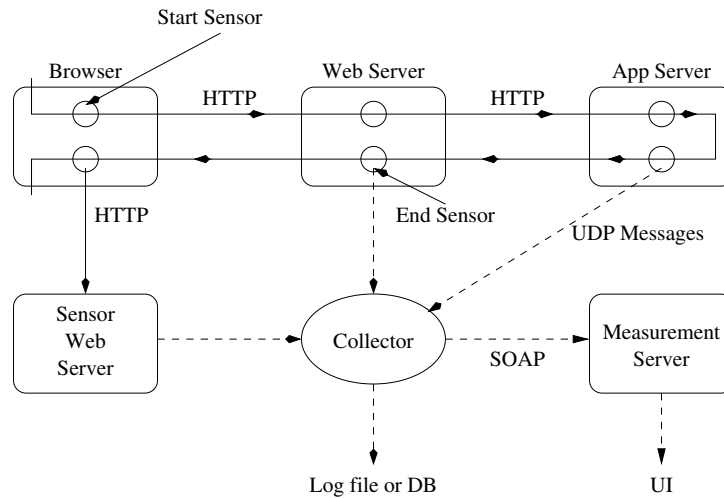


Figure 1. WebMon Architecture

Click Cookie Transmits the referrer page, and the time at which the user 'clicked' at the referrer page.

Load Cookie Transmits the time at which the requested page was loaded in the client's browser.

A sample client request as it is processed by the WebMon architecture is shown in Figure 2. Initially, the user browses through an already instrumented page *A*. This page is instrumented such that whenever the user clicks on a URI, a JavaScript function will record a timestamp of the form *timestamp : A* in the `click` cookie. Hence, when the user clicks on the URI for a page *B*, the appropriate `click` cookie is generated. The HTTP request is then sent to the web server. Since our web server is also instrumented, it records the timestamps before and after the processing of the request (t_2 and t_3 , respectively) and sends these times to the WebMon monitoring daemon along with the identifier t_1 (from the `click` cookie). The web server then sends its response back to the browser.

When the browser receives the requested page, it timestamps the end of the requests by storing the end time in the `load` cookie. Since this page is instrumented, we add a spurious request at the end, after setting the `load` cookie, for a '.wmi' page. Our instrumented web server understands that a '.wmi' request is a spurious request used solely for the purpose of instrumentation. It takes the client's start and stop times (t_1 and t_4 , respectively) from the `click` and `load` cookies and sends them off to the WebMon collector. The collector can correlate these times with that of the server because it uses t_1 as the identifier for the transaction.

Now we will consider the various situations in which parts of the infrastructure may not be instrumented.

An HTML page is not instrumented

If the referrer page is not instrumented the `click` cookie will not be set. The web server will report its data to the WebMon collector, using its own start time as the transaction identifier. When the requested page is loaded by the browser, it will realize that the `click` cookie is not set and will not send a '.wmi' request. Hence, we will not obtain any client-side data for this request.

If the requested page is not instrumented, then we will not get any data from the client side. The web server will still record its processing time and send it off to the WebMon collector, using its own start time as the transaction identifier.

An interesting situation arises if the referrer page is instrumented, the requested page is not instrumented, but a subsequently requested page is instrumented. In this situation, a simple `click` cookie approach will fail since the cookie will incorrectly monitor the start time of the click for the first page as the start time of the second request. To avoid this problem from happening, we include the referring page in the `click` cookie. Hence, the load function can check whether the `click` cookie is stale or not by comparing the referrer on the cookie with its own understanding of the referrer (through the history object).

The Web Server is not instrumented

If the web server is not instrumented, then we will not obtain any web server data. The client will do its own monitoring, and will send requests to the measurement server (the '.wmi' requests), which will store the client perceived

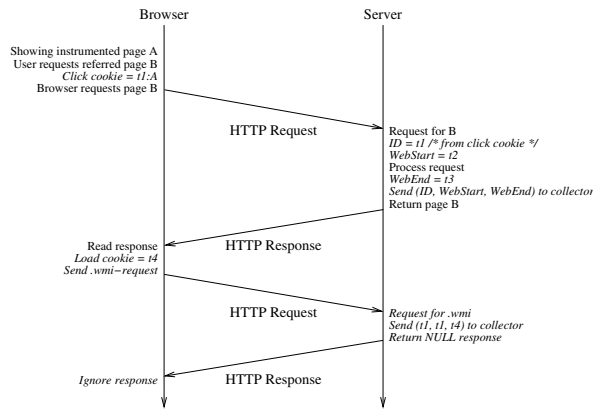


Figure 2. Monitoring Data Correlation

performance. Since the web server communicates with the application server using HTTP, the click cookie storing the request identifier is still forwarded to the application server.

3 Example Use

The data from the collector can be analyzed with respect to a variety of perspectives: business, transaction, client, server workload or a combination of each. Depending on the analysis this can be done online using an Interaction-Listener module or off-line. For the off-line data, the collector exports a log file in the following format:

```

972587010448 (/cgi-bin/browse.cgi)=\
{referrer=-1(http://source.hp1.hp.com/cgi-
bin/browse.cgi),\
client(15.4.90.244)=(-1,-1,-1),\
web(2130706433)=(972587400881,972587401603,722),
app(null)=(-1,-1,-1)}
  
```

The first part of the log entry gives the identifier of the particular request (972587010448) and its URL (/cgi-bin/browse.cgi). The second part gives the previous URL. The third part specifies the client's IP address (or the proxy) followed by a triplet (start, end, residence) times. Similarly we get triplets for the web and application server. In case any time is not available, its set to -1 (as for the application server in our case).

To evaluate WebMon, we ran an experiment by instrumenting an HP Intranet service, called Corporate Source [4]. The service is a two-tiered web service, running on Linux using an Apache web server with most of the functions offered by CGI scripts. The service does not have an application server component.

Figure 3 shows a graph of the requests for the Corporate Source server, with correlated values on an hourly basis for: (1) the number of requests, (2) the server response times,

and (3) the client response times. The response times are shown as minimum, maximum, and average values. The graph's y-axis is log scale. This is illustrative of the data available from WebMon. One can imagine trying to do this kind of correlation simply with a time based analysis of client and server logs. That approach, however, will have two limitations: (1) the client data will have to be monitored and obtained for several clients, and (2) in case the web server is reached through an HTTP redirection then the URL-based correlation will not work. Hence, WebMon presents a simpler alternative.

From the graph its clear that some transactions are taking a really long time (upto 60 minutes) for client response times, while average response times of 10 seconds is common. It's also clear that the web server response for these transactions is not extraordinarily poor. Hence, the performance bottleneck is either the network or the web service system that is handling the incoming TCP traffic. Our current prototype does not provide enough information to diagnose this further. We are currently extending the prototype to gather more information to enable such deductions to be made.

4 Related Work

The work in this paper relates to the work on performance monitoring of distributed applications in general, and Web performance monitoring in particular. Distributed systems performance monitoring tools have been built since the early eighties [13]. While the early tools could rely on a homogeneous infrastructure (e.g., all Unix machines on an Ethernet-based local area network), more modern tools accommodate the heterogeneity of the Internet (e.g., Vital-Sigs from Lucent).

The High Performance Computing community con-

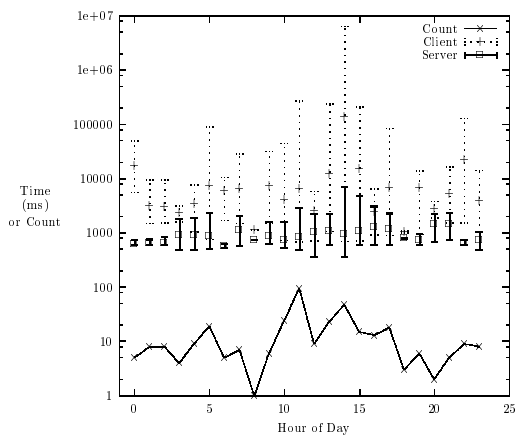


Figure 3. Example Monitored Data from WebMon

structs performance monitoring tools for distributed applications in order to tune the performance of large scale parallel and distributed applications. Since the infrastructure for such applications is mostly under the control of the designers of the applications, the monitoring tools that result from this work can assume an homogeneous infrastructure (e.g., Paradyn from University of Wisconsin [12], and Pablo from University of Illinois at Urbana-Champaign [15]).

Transactional use of the Internet has spawned a whole mini-industry on performance monitoring, characterization, and improvement of Internet-based transactions, e.g., VitalSigns, WebPartner, KeyNote, Akamai.

VitalSigns monitors Internet applications from an end-user's point of view [9]. They monitor various metrics such as the end-user perceived response times, the time spent in network and server components, the network bottleneck speed from the server to the client, and so forth. VitalSigns obtains these metrics from observing the TCP traffic at the client side. Since they do not put monitors in applications, they have to make various assumptions about the server and network behavior. VitalSigns has to extrapolate network delay from only the server to client path, it does not have information available for the client to server path. VitalSigns does not have a well-defined concept of a "session." Finally, VitalSigns information is made available to the end-user, but is not readily available to the web site administrator in the fashion of WebMon. Hence, although the two tools take a similar philosophical approach to web transactions monitoring (i.e., monitoring from an end-user's perspective) the metrics collected and the analysis enabled is quite different.

KeyNote (www.keynote.com) monitors web sites using HTTP probes that simulate user behavior. These probes can be scripted to go through typical transactions offered by the web site. KeyNote has a large number of geographically dispersed sites from which the probes can be de-

ployed. They compile availability and response time statistics from these probes. Since their probes are geographically dispersed, they can give a more accurate indication of locale-specific delays caused by Internet congestion.

WebPartner (www.webpartner.com) also monitors web sites using external HTTP probes. Their approach is similar to KeyNote, but they do not have a geographically dispersed probe infrastructure.

Mercury Interactive (www.mercuryinteractive.com) also uses geographically dispersed HTTP probes to monitor web sites. The probes are scripted by recording actual user behavior. Mercury Interactive also provides server side monitoring tools to help with the diagnosis of server side performance problems.

All the products mentioned above use HTTP probes, which simulate user activity. In that way, they all differ from our approach which enables the measurement response time as experienced by real users.

Candle [2] provides monitoring tools based on actual user experience, in an approach similar to the one described in this paper. The difference between their approach and ours is that they require a Java applet to be downloaded with each monitored page, whereas we only require a Javascript fragment to be included in each page. Moreover, Candle does not provide any means for correlating transactions at different levels (web browser, web server, application server) whereas we use cookies to enable this correlation.

Fu et al. [6] describe an approach for server-side monitoring of network traffic that allows determination of correlated client response times, network delays, and server processing times, for web transactions. Their approach does not require any modifications to the web service. Such network monitoring approaches, however, do not work with SSL traffic, as they rely on the ability to parse HTTP header information. Moreover, since the transactions are not identi-

fied, further breakdowns such as application server response times are not possible.

Websidestory (www.websidestory.com) and WebTrends (www.webtrends.com) also use client-side instrumentations using JavaScript to perform several client-side monitoring. Their main purpose, however, is to monitor client usage patterns for understanding their *business* implications. Hence, unlike WebMon, they don't monitor the response times and server response times for any given transactions, in addition to the usage patterns.

5 Conclusions

In this paper we have shown a monitoring architecture that provides data of all the components involved in a web transaction. Compared to a traditional approach where each component logs its own information to its own log file and making it difficult to correlate the various items, the WebMon architecture presents the data in a correlated form. We have described several different instrumentation techniques that enable such correlators to be passed between components of typical web services. We have demonstrated the use of the tool for monitoring an existing web service.

There are several directions for future work in this area:

1. We have found that instrumenting the different components of a web service is a challenging and time consuming task. Each component instrumentation in WebMon had to be developed separately, and then managed over time. Hence, any changes in the correlation would have to be manually propagated to the different component sensors. In the future, we'd like to use some generative programming techniques [3] to systematically develop component instrumentation from a specification.
2. The browser instrumentation currently does not work in the presence of HTML frames.
3. Our correlation scheme currently assumes a three-tiered architecture of a web client, web server, and an application server. In the future we will generalize this to an n-tiered system.
4. Our sensors currently do not support throttling, nor do they collect information other than response times. We will extend them in the future to support throttling and to collect information like CPU, disk, memory, network utilization, and so forth.
5. We want to add instrumentation between the browser and the server. This will enable us to better understand the performance situation as depicted in Figure 3, where the client response times are often quite high yet the server residence times are not so high.

References

- [1] R. Blake. *Optimizing Windows NT*. Microsoft Press, 1993.
- [2] Candle. Candle: ebusiness at the speed of light. See <http://www.candle.com>.
- [3] K. Czarnecki and U. W. Eisenecker. *Generative Programming - Methods, Tools, and Applications*. Addison-Wesley, June 2000.
- [4] J. Dinkelacker, P. Garg, R. Miller, and D. Nelson. Progressive Open Source. In *Proceedings of the 24th International Conference on Software Engineering*, Buenos Aires, Argentina, May 2002. To Appear.
- [5] F. ElRayes, J. Rolia, and R. Friedrich. The Performance Impact of Workload Characterization for Distributed Applications using ARM. In *Proceedings of the Computer Measurement Group (CMG) '98*, 1998.
- [6] Y. Fu, L. Cherkasova, W. Tang, and A. Vahdat. EtE: Passive End-to-End Internet Service Performance Monitoring. In *Proceedings of USENIX*, Monterey, CA, USA, June 10-15 2002.
- [7] P. K. Garg. An End-User's Perspective on Application Management. In *Proceedings of the 12th International Workshop on Distributed Systems Operations and Management (DSOM)*, Nancy, France, Oct. 2001.
- [8] T. O. Group. Systems Management: Application Response Measurement (ARM). See <http://www.opengroup.org/public/pubs/catalog/c807.htm>, 2000.
- [9] V. S. Inc. Characterizing End-to-End Performance. See <http://www.ins.com/knowledge/whitepapers/characterizing.asp>, 1999.
- [10] G. McDaniel. METRIC: A Kernel Instrumentation System for Distributed Environments. In *Proceedings of the Sixth ACM Symposium on Operating System Principles*, pages 93-99, Nov. 1977.
- [11] D. A. Menasce and V. A. F. Almeida. *Capacity Planning for Web Performance*. Prentice Hall, Englewood Cliffs, NJ 07632, 1998.
- [12] B. P. Miller, J. M. Cargille, R. B. Irvin, K. Kunchithap, M. D. Callaghan, J. K. Hollingsworth, K. L. Karavanic, and T. Newhall. The paradyn parallel performance measurement tools. *IEEE Computer*, 11(28), Nov. 1995.
- [13] B. P. Miller, C. Macrander, and S. Sechrest. A Distributed Programs Monitor for Berkeley UNIX. *Software-Practice and Experience*, 16(2):183-200, Feb. 1986.
- [14] J. Rolia and K. Sevcik. The Method of Layers. *IEEE Transactions on Software Engineering*, 21(8):689-700, Aug. 1995.
- [15] L. D. Rose, Y. Zhang, and D. Reed. Svpablo: A multi-language performance analysis system. In *Computer Performance Evaluation - Modelling Techniques and Tools. Springer-Verlag, (10th International Conference- Performance Tools'98)*, pages 352-355, Palma de Mallorca, Spain, Sept. 1998.
- [16] C. Woodside, J. Neilson, D. Petriu, and S. Majumdar. The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. *IEEE Transactions on Computers*, 44(1):20-34, Jan. 1995.