WALTy: A tool for Evaluating Web Application Performance

G. Ruffo, R. Schifanella, and M. Sereno
Dipartimento di Informatica
Università degli Studi di Torino
{ruffo,schifane,matteo}@di.unito.it

R. Politi
CSP Sc.a.r.l. c/o Villa Gualino
roberto.politi@csp.it

Abstract

In this paper we present WALTy (Web Application Load-based Testing tool), a set of tools that allows the performance analysis of web applications as they are perceived by the end users. The proposed approach is based on a workload characterization created from informations extracted from log files. The workload is generated by using of Customer Behavior Model Graphs (CBMGs) that allow to describe the behaviors of typical users on a web site.

1 A Framework based on Representative Workload Generation

Several approaches have been proposed for capacity planning and performance prediction: on one hand, analytical methods or simulation frameworks that works on a model of the system; on the other hand, we have benchmarking or load testing tools that operate on the real system or on a working test plant. WALTy is a powerful load testing tool written in Java that implements a *trace-based* stressing framework where traces are generated to describe the behavior of each *Virtual User* (VU) browsing the web site. In general, many stressing clients replicate a (set of) artificial session(s) or navigational patterns where sessions are randomly or manually generated, or extracted from log files. Our tool is implemented to provide every single VU a profile that is represented by means a Customer Behavior Model Graph [2]. This graph based formalism allows to describe the behaviors of typical users on the web application under test. During a test session, a synthetic workload is generated and performance related measures collected. In such scenario, WALTy uses the *httpperf* tool [3], developed at Hewlett-Packard, with particular attention respect to the load generator and statistics collector components. We have modified and integrated them in our tool. In the following, we briefly describe the fundamental modules of WALTy: (1) How CBMG are extracted from log files and (2) How CBMGs are used to generate virtual users behaviors.

2. CBMGBuilder: from log files to CBMGs

In this section we focus on the first component of WALTy: the creation of CBMGs from input data. Because a Customer Behavior Model Graph is a session-based representation of a user navigational pattern, session identification from data is a central topic, because web logs format is inherently hits-oriented. A wide range of *ad hoc* techniques have been implemented and adopted by the community to identify a web session [1, 4]: cookies, user authentication, URL rewriting, and so on.

WALTy implements a scalable methodology to session identification in order to handle with any new data source without modifying the core application or rewriting and recompiling the code, and integrate many different input formats in a common framework. Proposed methodology is based on the definition of an *abstract data model*, in such context called *CBMG Input Format (CIF)*, that includes any specific information necessary for the creation of CBMG.

Moreover, our tool introduces the concept of a *plug-in*. A plug-in performs the simple task of transforming the format of a generic source file into the data model cited above (e.g., CIF). When a new file format is encountered or a session identification technique is introduced, a new plug-in must be implemented. This process permits a simple and practical management of several session identification mechanisms, shifting the format conversion problem at implemented plug-ins. CBMGBuilder module is implemented merely using input file with CIF syntax and allow the creation, installation and deletion of a plug-in.

3. Generating Representative Web Traffic from CBMG

When a CBMG is generated for each user profile, the next step in the emulation process is traffic generation. As in the general *trace-based* framework, traffic is generated by means of a sequence of http-requests, with a think time between two successive requests. This Section describes how to generate such a sequence from CBMGs. Let us suppose

that the clustering phase returned m profiles $\{\Phi_i\}$, where $i = 1, \dots, m$. Each profile is a CBMG defined as a pair (P, Z) of matrices $n \times n$, where n is the number of states, P is the transition probabilities matrix and Z the server side think time matrix [2]. Observe that each profile Φ_i , corresponds to a set of sessions $\{S_{i1}, S_{i2}, \dots, S_{ip}\}$. Let us indicate the cardinality of this set of sessions with $|\Phi_i|$.

Moreover, let us define the *Representativeness* of profile Φ_i , as the value:

$$\rho(\Phi_i) = \frac{|\Phi_i|}{\sum_{i=1}^m (|\Phi_i|)},$$

which is the rate of the number of sessions corresponding to profile Φ_i , w.r.t., the total number of sessions.

When generating traffic to our system under test (SUT), profiles Φ_i are used to properly set up the behaviors of virtual users. The value $\rho(\Phi_i)$ gives a way to calculate a representative number of virtual users running with the same profiles. For example, if we want to start a test made of N virtual users accessing the server, we can parallelize stressing clients jobs as it follows: client i , that emulates sessions with profile Φ_i , runs $N * \rho(\Phi_i)$ virtual users, with $i = 1, \dots, m$.

WALTy allows further scalability: in fact, we can perform a fine-grained test, changing relative profiles percentage, e.g., we can run experiments responding to questions like “what does it happens when users with profile $\rho(\Phi_3)$ grows in number w.r.t. other classes of users?”. As a consequence, the framework can be generalized: client i runs $N * f_i * \rho(\Phi_i)$ virtual users, where $0 < f_i < 1$, if we want to reduce the representativeness of Φ_i , and $f_i > 1$, if we want to strengthen it. A further constraint is that the following condition must hold:

$$\sum_{i=1}^m f_i * \rho(\Phi_i) = 1.$$

Another interesting feature of WALTy, is given by the natural scalability of a profile characterization model. In fact, the analyst can perform a what-if analysis at transition level, changing values in matrices P and Z . For example, the analyst may be interested in the consequences of a navigational behavior alteration, e.g., if a new link is planned to be published in the home page, a different navigation of the occasional visitor is reasonably expectable.

Finally, we describe how a session can be generated from a CBMG. The procedure takes as input parameter a CBMG profile and the set $L = \{L_2, \dots, L_{n-1}\}$, where L_i is the list of objects (e.g., html files, cgi-bin, ...) belonging to the i -th state and returns a session, made of a sequence of http requests. The generation of a new session from a CBMG takes advance of the following steps:

- Given a state and the corresponding list of objects, the procedure selects an object (i.e., or simply a page) with a simple ranking criterion, i.e., pages that are frequently accessed, are likely to be selected. In other words, it is not a random choice, but it is a *popularity* driven page selection.
- In order to create a well formed httpperf request, WALTy associates the given page to the following set of properties: (i) the HTTP method (GET, HEAD, POST) should be selected page, (ii) in the case of a POST method, a byte-sequence to be sent to the server is allocated and, finally, (iii) if selected resource is a dynamic page (e.g. php script, jsp page, ...) needing a list of input parameters, WALTy will append to the request a sequence of (*name=value*) items.
- Then, WALTy selects the next state to be visited during the session. The random selection is weighted by means of transition probabilities contained in P .
- A think time extracted from matrix Z is defined in the request. Observe that, by default, the web logs give timestamps that allow to obtain the distance from two consecutive client requests. Moreover, the informations stored in web logs are inherently insufficient to calculate the exact value of client think time without modifying completely the client implementation to track this value. For this reason, the solution adopted in our tool uses only the informations available from web log files, i.e. the value of server side think time extracted from matrix Z .

Acknowledgments

This work is partially supported by the Italian minister of Research (MIUR), within the project *WebMinds* (FIRB).

References

- [1] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1(1):5–32, 1999.
- [2] D. Menascé, V. A. Almeida, R. Fonseca, and M. A. Mendes. A methodology for workload characterization of e-commerce sites. In *Proc. of ACM Conf. on E-Commerce*, Nov 1999.
- [3] D. Mosberger and T. Jin. httpperf: A Tool for Measuring Web Server Performance. In *In Proc. of First Workshop on Internet Server Performance*, pages 59–67. ACM, 1998.
- [4] J. Pitkow. In search of reliable usage data on the WWW. *Computer Networks and ISDN Systems*, 29(8–13):1343–1355, 1997.