Geist: A Web Traffic Generation Tool

K. Kant, V. Tewari and R. Iyer

Intel Corporation, Hillsboro, OR
{krishna.kant|vijay.tewari|ravishankar.iyer}@.intel.com

Abstract. This paper briefly describes the design of Geist, a traffic generator for stress testing of web/e-commerce servers. The generator provides a large number of dialable parameters that allow traffic characteristics to range from simple static web-page browsing to the transactional traffic seen by e-commerce front end servers. Unlike other traffic generators, Geist concentrates on the characteristics of the aggregate traffic arriving at the server, which allows for realistic traffic generation without any explicit modeling of users or network components.

1 Introduction

The work reported here was motivated by the need to understand the implications of web traffic characteristics on the architectural aspects of a web/e-commerce server. This requires generation of the aggregate Internet traffic from a server's perspective without having to explicitly emulate users, network components, or network protocols. This *server-centric* focus also means that client-level aspects of the traffic are relevant only to the extent they affect the aggregate traffic. Furthermore, lab testing environment often requires that the servers be driven to their capacity (and in fact beyond the capacity into overload region). This requires the capability to generate heavy traffic (e.g., more than 10K requests per second). It has been amply noted in the literature that the traditional socket interface makes it very difficult to overload web-servers [10]. Finally, the increasing dynamic content of web pages makes generation of realistic transactional characteristics particularly challenging, especially in e-commerce environments. In this paper, we present a tool called Geist (Generator of E-Commerce and Internet Server Traffic), that has been designed to deal with these and other important issues.

The aggregate traffic generation approach used by Geist allows for a more scalable traffic generation since the request-send and response-receive operations on the client side can be completely decoupled. In particular, it is possible to keep generating successive requests at prescribed times irrespective of how long the server takes to send back the responses. Furthermore, the traffic impact of network components and protocols (e.g., queuing delays, flow-control, retransmission, rerouting, etc.) can be captured without having to explicitly represent the network or the protocols. In effect, the approach converts the inherently "closed loop" traffic characterization/handling problem to an "open loop" problem where the traffic properties are modeled directly at the server. The main disadvantage of this approach is the difficulty in directly modeling the user behavior. Geist handles this by carving out a set of "virtual users" from the aggregate traffic as discussed later.

Generation of aggregate traffic with complex characteristics is often computationally intensive, which makes it difficult to ensure that a request is actually generated very close to the intended time. Geist addresses this issue by splitting the generation into two steps referred to as *trace generation* and *traffic generation* respectively. Trace generation handles all the complexities of computing the actual time and parameters for the requests, whereas the traffic generation step simply reads the trace and issues the requests. The added advantage of this approach is that the trace could very well have been derived from HTTP logs from a live site. The traffic generator part still has to deal with several O/S related issues to ensure accurate timing of the generated traffic. These issues are discussed in Section 3.

In related work, some other well known tools are Microsoft's Web Application Stress Tool, HTTP-perf [10] from HP Labs, and SURGE [1] from Boston University. These tools mostly take the user emulation approach whereas Geist attempts to generate aggregate traffic directly. The first two tools are very limited in terms of the temporal properties that the traffic is allowed to have. SURGE, on the other hand, supports self-similarity by making each "user equivalent" an on-off process. In contrast, Geist supports asymptotically self-similar, multifractal, and non-stationary arrival process. However, Geist is not locked into producing self-similar traffic; it can also generate traffic with much shorter dependencies, including none. Geist also supports detailed transactional characterization of the traffic. However, SURGE does explicitly support embedded requests, a feature, that is currently not implemented in Geist. Geist's model for temporal locality in accesses is taken from SPECweb96/99 benchmarks, rather than emulated explicitly as in SURGE. Unlike the current version of Geist, the HTTP-perf tool supports multiple requests per connection and cookies; however, it is designed basically for deterministic inter-request time on a single client. Of the above tools, only Geist and WAST can generate a mix of non-secure and secure traffic. Geist also allows for complex temporal characteristics including long-range dependence, traffic irregularities at intermediate time scales, and nonstationarity.

2 Traffic Properties Considered in Geist

In this section, we briefly discuss the major traffic properties considered in Geist and how they are introduced. A more detailed discussion of Geist's design may be found in [8].

Temporal Distribution of the traffic: The temporal properties of the traffic can be examined either in terms of inter-arrival times or the arrival counting process (i.e., number of arrivals in a "slot" of certain duration). The latter is usually more robust, and is used in Geist. An appropriate slot duration is the time-period over which the average number of arrivals are in the range of a few 10s. Geist allows control over the first two moments of the arrival counting process as discussed later in this section. The arrivals belonging to a slot are distributed uniformly so that the inter-arrival time distribution within a slot is negative exponential.

User Behavior Impact on Temporal Properties of the traffic: Apart from the distributional properties, the correlation structure of the arrival process is crucial from a temporal characterization point of view. Over large time scales, these correlational properties

are a result of the user behavior, which is often characterized by low and high activity periods, each with a heavy-tailed distribution. The aggregate traffic in such cases exhibits long-range dependence or *self-similarity* [11]. Although self-similarity is now established as a universal property of high-speed networks, most of the studies in the literature take a *network centric view* where the traffic metric of interest is the bytes traveling over a link. In contrast, ours is a *server centric view*, where the interest lies in the requests coming into the server. Our extensive analysis of web and e-commerce request traffic [4, 7] confirms that the self-similarity is observed at the request level as well.

The correlations in the arrivals process (and, in particular, the self-similarity) can substantially affect the queuing behavior of the traffic depending on the time-scales captured by the queue. Since we do not know a priori what time-scales are of interest, it is crucial that the traffic generator be able to generate self-similar traffic, if needed. Geist provides this capability. Geist also supports generation of traffic with medium and short range correlations, including no correlations at all (i.e., generation of a renewal process).

Geist controls the temporal properties of the traffic by using the M/G/∞ paradigm that can generate asymptotically self-similar as well as short and medium range dependent traffic. As such, the marginal distribution of the generated arrival process is Poisson (the occupancy distribution in a M/G/∞ queue), however, it can be easily transformed to a process with the desired distribution as detailed in [9]. Such a transformation is adequate at least as far as second order properties are concerned.

Network Dynamics Effects on the traffic: As the traffic passes through the network elements, its characteristics are affected not only by the usual queuing/processing delays at the nodes but also by network level mechanisms such as TCP flow and congestion control and segmentation/reassembly of packets at intermediate nodes. The time constants of these activities usually fall in 100 ms range which is much less than the time constants involved in user activities. Consequently, self-similarity aspects of the traffic are typically not affected by the network dynamics but network dynamics has a substantial influence on the queuing performance of the traffic.

A direct emulation of network effects on the traffic makes sense only for network-centric studies; with server-centric studies that we are interested in, representation of network effects would require too much additional machinery. Thus an indirect representation of these effects in the aggregate traffic is ideal for our purposes. It has been shown recently that these effects can be captured by using the concept of *multifractal* properties which are defined as an extension of self-similarity [2]. As with self-similarity, much of the work to date on this aspect considers byte traffic on a network link, however, it is likely that similar properties apply at the request level also. Geist provides the capability of introducing multifractal-like properties in the traffic in case such effects are deemed important. This is done by accumulating arrivals over some 2^K time-slots and using a *semi-random cascade generator* to distribute the mass down to individual slots. More details on this may be found in [6].

Traffic Nonstationarity: Much of the work on web-traffic analysis assumes that the traffic is approximately stationary, which has been confirmed by our own analysis of web traffic from several sources. However, our recent analysis of traffic logs from sev-

eral business to business (B2B) and business to consumer (B2C) sites shows that even over short intervals of 10-15 minutes, the traffic is often nonstationary [7]. Geist provides the capability to introduce nonstationarity in the traffic should this be important in studies such as dynamic resource-provisioning.

In order to avoid any perturbation to the M/G/∞ paradigm, the nonstationarity is introduced at the output end by modulating the number of arrivals during the last slot given by the M/G/∞ process. This is done using a level-shift process, the details of which may be found in [7].

Transactional Composition of the traffic: Representation of transactional behavior is another important aspect of e-commerce traffic generation. Basically, this involves a classification of the transactions based on the server-side scripts invoked by a request which in turn invoke middleware (e.g., shopping cart, search server) or backend functions (e.g., product availability, pricing, etc.). The primary motivation for this classification is that different scripts may have very different characteristics in terms of CPU and I/O requirements. Another important aspect of transactional composition is the use of secure HTTP (or HTTPS) for some (possibly all) transactions. As shown in [5], an HTTPS is far more expensive than a corresponding HTTP transaction, thereby making the distinction necessary.

The transactional characteristics can be easily represented using a "state machine" or a Markov chain, where the state represents the transaction type and the transitions represent ordering between them. In Geist, each state could be further classified as "secure" or "nonsecure" and is associated with a server side script.

It is important to note that the transactional classification applies to individual users, whereas Geist generates only the aggregate traffic. Therefore, the generated traffic needs to be split up into requests for an individual "virtual" user. Geist does this as follows: First it computes the number of virtual users K as the product of the aggregate mean arrival rate and the inter-request time for an individual user, both of which are input parameters to Geist. The successive arrivals are then marked equiprobably to belong to one of the K user streams. Each user stream is then assigned a unique-id and the transactions are marked according to the Markov chain model.

Access Characteristics of the traffic: The actual server-side script executed by a transaction embodies the resource requirements in processing the transaction. Geist only indicates which scripts are executed and how often; it does not explicitly specify the properties of these scripts or even their parameters. Such an approach gives maximum flexibility to the experimenter in writing the scripts (or simply using the existing scripts for the e-commerce application of interest). The parameter issue can be resolved by writing a wrapper script that generates the desired parameter values. We plan to provide some sample wrapper scripts along with the tool.

Irrespective of the type of access (dynamic or static), it is important to control the characteristics of the response sent out by the server. This is done in Geist by requiring that every Get request retrieve a "file" from a given "file-set". As in the SPECweb99 benchmark, it is assumed that a dynamic GET simply creates some additional data (assumed to be rather small in size), which is appended or prepended to the static file being requested. The details of file-set and file-access specification are similar to those in SPECweb96 and SPECweb99 benchmarks (`www.specbench.org/osg/`). In fact,

Geist's approach is somewhat more general and can easily emulate access characteristics of both of these benchmarks.

3 Geist Capabilities and Traffic Generation Issues

Geist trace generator defines 30 or so control parameters to which values can be assigned in an input file. The generated trace is written to an output file, which can be further split up into multiple files for use by each client in the traffic generator part. The input file can contain 3 types of statements: (1) declaration of auxiliary variables (scalars or arrays), (2) assignments of expressions to scalar or array variables (predefined or declared in the input file), (3) Miscellaneous controls (diagnostic, pause, change values and resume, etc.). Currently, there is no graphics interface for the input, nor is one planned.

In addition to trace generation, the generator also includes the following capabilities: (1) Simulation of a simple single-server queuing system directly driven by the generated traffic, (2) Emulation of a file-cache also directly driven by the generated traffic, (3) Creation of the file-system that will be exercised by the traffic generator, and (4) Automated analysis of the temporal properties of the generated traffic. These capabilities allow study of various properties of the generated traffic in a convenient way. The entire trace generator is written in C and runs on both Unix and Windows environments. It should be relatively straightforward to port it to other environments as well.

The traffic generator part can use multiple (say, N) client boxes, in which case the requests in the trace file are assigned to each client such that (a) all requests issues by a "virtual user" stay on the same client, and (b) load is spread evenly across all clients. This splitting of the original trace file is done offline in order to avoid any timing issues. Each client runs an instance of the Geist traffic generation engine which includes a pool of (request) sender threads, (response) receiver threads and a timer thread (to watch run duration). One of the clients called *prime client* is responsible for sending global information to the other client boxes including IP address of the server, duration of the test, number of sender/receiver threads, etc. Additionally the prime client performs time synchronization amongst all clients by broadcasting a time interval after which all the clients reset their virtual clocks to zero. All time decisions are based on this time zero from here on. After performing these functions the prime client behaves like any other client box. Like the trace generator, the behavior of traffic generator is controlled by assigning values to a set of predefined parameters in an input file.

The current request generation engine has been implemented in the C language on the Win32 platform. In this implementation, the communication between the prime client and the other clients is based on UDP broadcast. This restricts the implementation to having all clients in the same subnet. A more flexible implementation including the integration of *pthreads library* is planned for the future.

In order to avoid timing skew due to I/O delays, the input file created by the trace generated is locked in the memory at the beginning. Secure requests are supported via the OpenSSL library (see `www.openssl.org`). Geist also supports secure and non-secure requests via proxies. Furthermore, it is possible to add custom headers in the configuration file which will be used at the time of making the request. This makes the architecture extensible to support new HTTP request headers.

An important issue for the traffic generator is to ensure that the temporal properties of the generated requests are close to those of the traffic specified by the trace file. A logging utility has been provided for this purpose. A study of the accuracy of the generated traffic may be found in [8].

4 Uses and availability of Geist

We have used Geist already in generating traffic for a number of projects including a study of overload control strategies for web servers [3] a proxy server study and architectural studies like SSL performance implications [5]. Reference [3] uses GEIST to show the impact of loading a web server beyond its capacity. One of the advantages of using Geist (as opposed other traffic generators like Microsoft's WAS tool) here was that the traffic generated by the clients was not dependent on the implicit feedback of the varying load on the server. Geist was modified to receive the overload feedback (via a UDP channel) and respond by throttling the inter-arrival rate of the outgoing traffic. This further shows the flexibility of Geist's architecture for performing studies that require dynamic source traffic modifications based on the feedback.

Geist is to be regarded as work in progress, and can be enhanced in many ways. In particular, Geist currently does not support the new features of HTTP 1.1 or the detailed specification of scripts executed by the server. Our ongoing work shall address these and related issues. The entire tool along with the documentation can be downloaded from `kkant.ccwebhost.com/download.html`.

References

1. P. Barford, and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", Proc. ACM Sigmetrics, pp. 151-160, July 1998.
2. A. Feldmann, A.C. Gilbert and W. Willinger, "Data Networks as Cascades: Investigating the multifractal nature of Internet WAN traffic", Proc. ACM Sigcomm 98, pp42-55.
3. R. Iyer, V. Tewari and K. Kant, "Overload Control Mechanisms for Web Servers", Performance and QoS of Next Generation Networks, Naogya, Japan, Nov 2000.
4. K. Kant and Y. Won, "Server Capacity Planning for Web Traffic Workload", IEEE transactions on knowledge and data engineering, Oct 1999. pp731-747.
5. K. Kant, R. Iyer and P. Mohapatra, "Architectural Impact of Secure Socket Layer on Internet Servers", Proc. of ICCD, Sept 2000.
6. K. Kant, "On Aggregate Traffic Generation with Multifractal Properties", Proc. of GLOBECOM'99, Rio de Janeiro, Brazil, pp 1179-1183.
7. K. Kant and M. Venkatachelam, "Modeling traffic non-stationarity in e-commerce servers", available at `kkant.ccwebhost.com/download.htm`.
8. K. Kant, V. Tewari and R. Iyer, "Geist: A generator of e-commerce and internet server traffic", Proc. of ISPASS, Tucson, AZ, Nov 2001, pp 49-56.
9. M. Krunz and A. Makowski, "A source model for VBR Video traffic based on M/G/∞ Input", Tech. Report, Univ of Maryland.
10. D. Mosberger and T. Jin, "HTTPPERF: Atool for measuring web server performance", Technical Report, HP Labs, 1998.
11. W. Willinger, M.S. Taqqu, R. Sherman, and D.V. Wilson, "Self-Similarity through High Variability: Statistical Analysis of Ethernet LAN traffic at the source level", Proc. of Sigcomm 95, pp100-113.