Benchmarking Models and Tools for Distributed Web-Server Systems

Mauro Andreolini[1], Valeria Cardellini[1], and Michele Colajanni[2]

[1] Dept. of Computer, Systems and Production
University of Roma "Tor Vergata"
Roma I-00133, Italy
{andreolini,cardellini}@ing.uniroma2.it,
[2] Dept. of Information Engineering
University of Modena
Modena I-41100, Italy
colajanni@unimo.it

Abstract. This tutorial reviews benchmarking tools and techniques that can be used to evaluate the performance and scalability of highly accessed Web-server systems. The focus is on design and testing of locally and geographically distributed architectures where the performance evaluation is obtained through workload generators and analyzers in a laboratory environment. The tutorial identifies the qualities and issues of existing tools with respect to the main features that characterize a benchmarking tool (workload representation, load generation, data collection, output analysis and report) and their applicability to the analysis of distributed Web-server systems.

1 Introduction

The explosive growth in size and usage of the Web is causing enormous strain on users, network service, and content providers. Sophisticated software components have been implemented for the provision of critical services through the Web. Consequently, many research efforts have been directed toward improving the performance of Web-based services through caching and replication solutions. A large variety of novel content delivery architectures, such as distributed Web-server systems, cooperative proxy systems, and content distribution networks have been proposed and implemented [35].

One of the key issues is the evaluation of the performance and scalability of these systems under realistic workload conditions. In this tutorial, we focus on the use of benchmarking models and tools during the design, testing, and alternative comparison of locally and geographically distributed systems for highly accessed Web sites. We discuss the properties that should be provided by a benchmarking tool in terms of various parameters: applicability to distributed Web-server systems, realism of workload and significance of the output results. The analysis is also influenced by the availability of the source code and the customizability of the workload model. We analyze popular products that are

M.C. Calzarossa and S. Tucci (Eds.): Performance 2002, LNCS 2459, pp. 208–235, 2002.
© Springer-Verlag Berlin Heidelberg 2002

free or at nominal costs, and provide source code: httperf [32], SPECweb99 (including the version supporting SSL encryption/decryption) [38,39], SURGE [7, 8], S-Clients [6], TPC-W [41], WebBench [45], Web Polygraph [42], and WebStone [30]. For this reason, we do not consider commercial tools (e.g., Technovations' Websizr [40], Neal Nelson's Web Server Benchmark [34]) that are more expensive and typically unavailable to the academic community, although they provide richer functionalities. Other benchmarking tools that come from the research (e.g., Flintstone [15], WAGON [24]) have not been included because they are not publicly available.

We can anticipate that none of the observed tools is specifically oriented to testing distributed Web-server systems, and only a minority of them reproduces the load imposed by a modern user session. Many existing benchmarks prefer to test the maximum capacity of a Web server by requesting objects as quickly as possible or at a constant rate. Others with more realistic reproductions of user session behavior (involving multiple requests for Web pages separated by think times) refer to request and delivery of static content only. This result was rather surprising if we think that the variety and complexity of offered Web-based services require system structures that are quite different from the typical browser/server solutions of the early days of the Web. The increasing need for dynamic request, multimedia services, e-commerce transactions, and security are typically based on multi-tier distributed systems. These novel architectures have really complicated the user and client interactions with a Web system, ranging from simple browsing to elaborated sessions involving queries to application and database servers. Not to say about the manipulations to which a user request can be subject, from cookie-based identifications to tunneling, caching, and redirections. Moreover, an increasing amount of Web services and content are subject to security restrictions and secure communication channels involving strong authentication that is becoming a common practice in the e-business world. Since distributed Web-server systems typically provide dynamic and secure services, a modern benchmarking tool should model and monitor the complex interactions occurring between clients and servers. None of them seems publicly available to the academic community.

We illustrate in Fig. 1 the basic structure of a benchmark tool for distributed Web-server systems that we assume based on six main components (benchmarking goal and scope, workload characterization, content mapping on servers, workload generation, data collection, data analysis and report) that will be analyzed in details in the following sections. The clear identification of the characteristics to be evaluated is at the basis of any serious benchmarking study that cannot expect to achieve multiple goals. From this choice, the *workload representation* phase takes as its input the set of parameters representing a given workload configuration and produces a non ambiguous Web workload specification. In the case of a distributed Web-server system, the content is not always replicated among all the servers, hence it is important that the *content mapping* phase decides the assignment of the Web content among multiple front-end and back-end servers. The *workload generation engine* of a benchmark analyzes the workload specifi-

cation and produces the offered Web workload, issuing the necessary amount of requests to the Web system and handling the server responses. The component responsible for *data collection* considers the metrics of interest that have been chosen in the first phase of the benchmarking study and stores relative data measurements. Often, the whole set of measurements must be aggregated and processed in order to present meaningful results to the benchmark user. The *output analysis and report* component of a benchmark takes the collected data set, computes the desired statistics, and presents them to the benchmark user in a readable form.

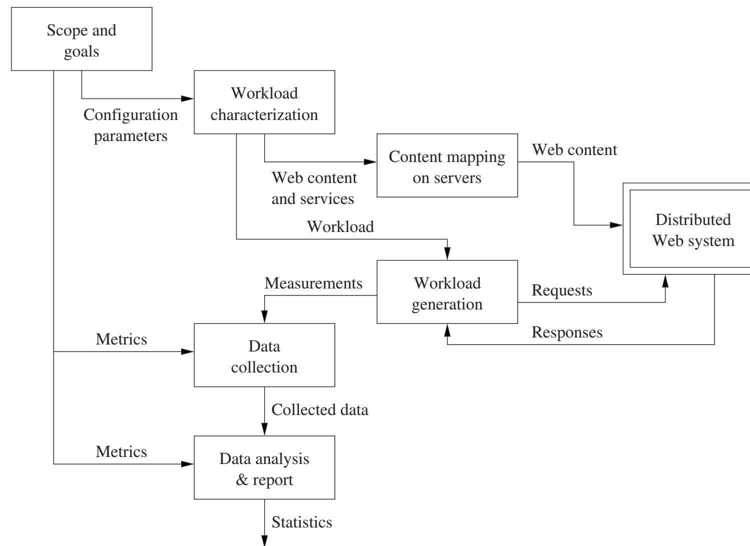

Fig. 1. Main components of a benchmarking tool for distributed Web-server systems.

After a brief description in Sect. 2 of the main architectures for locally and geographically distributed Web-server systems, the remaining sections of this tutorial follows the components outlined in Fig. 1. Finally, Sect. 9 concludes the paper and summarizes some open issues for future research.

2 Distributed Web-Server Systems

In this section we outline the main characteristics of the Web-server systems we consider in this tutorial, by distinguishing *locally* from *geographically* distributed architectures.

Any distributed Web-server system needs to appear as one host to the outside world, so that users need not be concerned about the names or locations of the replicated servers. Although a large system may consist of dozens of nodes, it is

publicized with one site name to provide a single interface to users at least at the site name level.

2.1 Locally Distributed Architectures

A locally distributed Web-server system, namely *Web cluster*, is composed by a multi-tier architecture placed at a single location. A typical architecture is shown in Fig. 2. A modern Web cluster has typically a front-end component (called *Web switch*) that is located between the Internet and the first tier of Web server nodes, and it acts as a network representative for the Web site. The Web system comprises also one authoritative Domain Name System (DNS) server for translating the Web site name into one IP address. The role of this name server is easy because a Web cluster provides to the external world a single virtual IP address that corresponds to the IP address of the Web switch.

The HTTP processes running on the Web server nodes listen on some network port for the client requests assigned by the Web switch, prepare the content requested by the clients, send the response back to the clients or to the Web switch depending on the cluster architecture, and finally return to the listen status. The Web server nodes are capable of handling requests for static content, whereas they forward requests for dynamic content to other processes that are interposed between the Web servers and the back-end servers. In less complex architectures these middle-tier processes (e.g., CGI, ASP, JSP) are executed on the same nodes where the HTTP processes run, so to avoid a connection with another server node. These middle-tier processes are activated by and accept requests from the HTTP processes. They interact with database servers or other legacy applications running on the back-end server nodes for providing dynamic content.

In Fig. 2 we evidence the three main flows of interactions of a client with the Web cluster not including secure connections: requests for static files that are served from the disk cache of the Web servers, requests for static files that require the disk access, requests for dynamic content.

The Web switch receives the totality of inbound packets and distributes them among the Web servers. The two main architecture alternatives can be broadly classified according to the OSI protocol stack layer at which the Web switch operates the request assignment, that is *layer-4* and *layer-7* Web switches. The main difference is the kind of information available to the Web switch to perform assignment and routing decision.

Layer-4 Web switches work at TCP/IP layer. They are *content information blind*, because they determine the target server when the client establishes the TCP connection, before sending out the HTTP request. Therefore, the type of information regarding the client is limited to that contained in TCP/IP packets, that is IP source address, TCP port numbers, SYN/FIN flags in the TCP header.

Layer-7 Web switches work at the application layer. They can deploy *content-based* request distribution. The Web switch establishes a complete TCP connection with the client, inspects the HTTP request content, and then relays it to

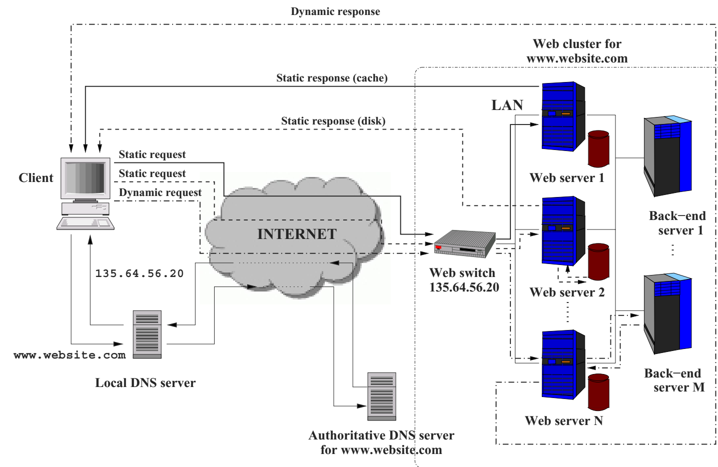

Fig. 2. Flows of interaction in a locally distributed architecture.

the chosen Web server. The selection of the target server can be based on the Web service/content requested, as URL content, SSL identifiers, and cookies.

Another important classification regards the mechanism used by the Web cluster to route outbound packets to the clients. In *two-ways* architectures, both inbound and outbound traffic pass through the Web switch. In *one-way* architectures, only inbound packets flow through the Web switch, while outbound packets use a separate high-bandwidth network connection. A detailed description of request routing mechanisms and dispatching algorithms for locally distributed architectures can be found in [10].

2.2 Geographically Distributed Architectures

A locally distributed system is a powerful and robust architecture from the server point of view, but does not solve the problems related to network delivery, such as first and last mile connectivity, router overload, peering points. An alternative solution is to distribute the server nodes over the Internet. With respect to clusters of nodes that reside at a single location, geographically distributed Web-server systems can reduce network delays experienced by the client, and also provide high availability to face network failures and congestion.

For performance and availability reasons, the distribution take typically place at the granularity of Web clusters that is, each geographically distributed node consists of a cluster of servers as that described in the previous section. We refer to this architectures as to *Web multi-cluster*. It maintains one hostname for the extern as in the Web cluster case, but now each Web cluster has a visible IP address. Hence, the request assignment process can occur in two or more steps. The first request assignment (*inter-cluster*) is typically carried out by the authoritative Domain Name Server (DNS) of the Web site that selects the IP

address of the target Web cluster during the address lookup of the client request. The second (*intra-cluster*) dispatching level is executed by the Web switch of the target cluster that distributes the request reaching the cluster among the local Web server nodes. A third (*extra-cluster*) dispatching level based on some request re-routing technique may be integrated with the previous two mechanisms [11, 35].

3 Scope and Goals of the Benchmarking Study

In considering the performance of a Web system we should regard to its software, operating system, and hardware environment, because each of these factors can dramatically influence the results. In a distributed Web-server system, this environment is further complicated by the presence of multiple components, that require connection handoffs, process activations and request dispatching. For example, referring to the Web switch component in Fig. 2, we may be interested to evaluate several alternatives, such as hardware, operating system, network related software, request dispatching policy and request forwarding mechanism. A Web server is characterized by similar hardware and software layers, and besides them by the HTTP software, the data distribution, the software for dynamic requests. A back-end server is also characterized by application and database software. Not to say of the additional complexity that characterizes a geographically distributed system.

A complete performance evaluation of all layers and components of a distributed Web-server system is simply impossible. Hence, any serious benchmarking study should clearly define its goals and limit the scope of the alternatives to be considered. In particular, this tutorial focuses mainly on benchmarking tools used in the design and prototype phase when different architectures must be evaluated and alternative solutions must be compared through experiments in a laboratory. Our main interests do not go to the hardware and operating system that in most cases are simply given. Similarly, we are not interested to evaluate the end-to-end performance of an installed Web system although many considerations can be also used for these purposes.

4 Workload Characterization

The characterization of the workload generated by a Web benchmarking tool represents a central aspect of benchmarking and constitutes a distinguishing core feature of existing tools as on it founds the attempt to mimic the real-world traffic patterns observed by Web-server systems. The generation of synthetic Web traffic is not a trivial task because it aims at reproducing as accurately as possible the characteristics of real traffic patterns, which exhibit some unusual features such as burstiness and self-similarity [4,12]. On the other hand, real world workloads are inherently irreproducible, since it is impossible to replicate the overall conditions under which the performance testing was originally performed.

In this section, we identify the main properties that are at the basis of the process of specifying the workload characterization. Moreover, we analyze the requirements that are specific for the benchmarking of distributed Web-server systems, compare the identified approaches, and discuss how the existing benchmarks realize these properties, providing also directions which we feel should be considered in the realization of benchmarking tools specific to distributed Web-server systems.

4.1 Classification of Alternatives

The workload characterization of a Web benchmark deals with three main aspects:

- the *Web service characterization* defines the types of services requested to the Web-server system;
- the *request stream characterization* defines the characteristics and the methodology used to generate the stream of requests issued to the Web-server system under evaluation;
- the *Web client characterization* defines the behavioral model of the Web client (i.e., the browser) and specifies to which extent the client characteristics support the HTTP specifications.

Characterization of Web-based Services. Let us first examine the characterization of Web-based services. As the variety of services and functions offered over the Web is steadily increasing, and puts dramatic performance demands on Web servers, the workload characterization of a benchmark should attempt to model realistic Web traffic and aim to capture this large variety of services. That is to say, the requests cannot be limited to static resources, but rather the workload should at least include dynamic services, which typically impose higher resource demands on Web servers [2]. Streaming multimedia services provided over the Web are also becoming increasingly popular and should be taken into account in the workload model. Security is a further issue which is often neglected in existing Web server benchmarks. With the increasing number of sensible and private transactions being conducted on the Web, security has raised its importance; therefore, modern workload characteristics should also include encrypted client-server interactions.

In Table 1 we summarize the core parameters that are involved in the specification of the offered workload. The definition of the parameters is oriented to the user session and resembles that described in [7,8,23]. The first set of parameters reviews some basic terminology, the second contains user-oriented parameters, while the third concerns Web object characteristics.

Characterization of the Request Stream. There are several possibilities to generate the stream of Web requests that will reach the tested system. The choice of a methodology impacts on the characteristics of the offered Web workload as

Table 1. Main parameters involved in the specification of Web workload.

Name	*Meaning*
Web page	A collection of objects constituting a multipart document intended to be rendered simultaneously; the base object is the first fetched from the server, then it is parsed, and all embedded objects are subsequently requested
User session	A sequence of requests for Web pages (clicks) issued by the same user during an entire visit to the Web site
Session length	The number of Web pages constituting a user session
Session interarrival rate	The rate at which new user sessions are generated
User think time	The time between two consecutive Web pages retrievals
Object sizes	The size of the collection of objects stored on the Web system
Request sizes	The size of objects transferred from the Web system
Object popularity	The relative frequency of requests made to individual objects
Embedded objects	The number of objects (not counting the base object) composing a single Web page
Temporal locality	How likely a requested object will be requested again in the near future

well as on the mapping of the synthetic content on the Web-server system that will be analyzed in Sect. 5. As shown in Fig. 3, the generation of the stream of Web requests falls into main four approaches.

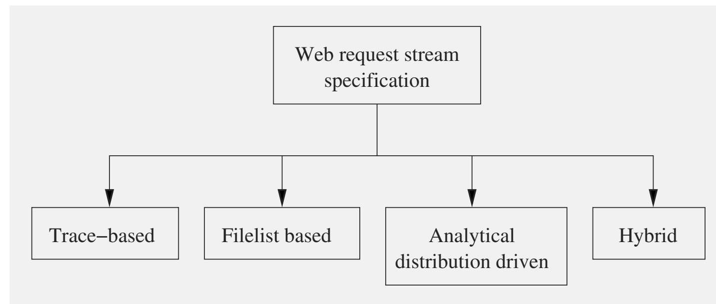

Fig. 3. Possible approaches to generate the stream of Web requests.

In the *trace-based* approach, the characteristics of the Web workload is based on pre-recorded (or synthetically generated) trace logs derived by server access logs [20]. The workload characteristics can be reproduced by replaying (or sampling) the requests as logged in the trace. An alternative is to create an abstract model of the Web site and extract session-oriented high-level information (such as session lengths and inter-arrival times) through a preliminary trace analysis that pre-processes server logs [25]. Some techniques to infer Web session characteristics from trace logs have been described in [1,27]. The trace-based approach

allows the benchmark tool to mimic the user behavior in a realistic way. However, the conclusions drawn from the experiments depends on the trace representativeness, as a trace can present workload properties that are strictly peculiar to it and do not have general validity. Furthermore, it can be hard to adjust the workload to imitate future conditions or varying demands.

It should also be remarked that, unlike the early days of the Web, server access logfiles are becoming a precious source of business and marketing information. As a consequence, companies and organizations are not willing to give their traces for free (or even at all), if not after years when the realism of these traces is at least doubtful. A further issue of the trace-based approach regards the reconstruction of the user sessions from the trace logs, which is not a trivial task [1]. For example, as sessions are identified through their IP address, it may happen that clients behind the same proxy are considered as coming from the same machine, which may lead to an improper characterization of the Web workload. Another issue that may complicate the reconstruction of user sessions, especially for highly accessed Web systems, concerns the coarse time resolution at which requests are recorded in server access logs [20].

In the *filelist based* approach, the tool provides a list of Web objects with their access frequencies. The object sizes are typically based upon the analysis of logs from several Web sites. During the workload generation phase, the next object to be retrieved is chosen on the basis of its access frequency. Time characteristics are typically not taken into account, hence the stream of requests depends only on the filelist while the inter-arrival request time is set. The filelist approach lacks of flexibility with respect to the workload specification, and also ignores the concept of user sessions As discussed in [4,3,8,12], Web traffic is bursty, session-oriented, and characterized by heavy-tailed distributions, which have high or even infinite variance and therefore show extreme variability on all time scales. To emulate these workload characteristics, it is not sufficient to mimic the user activity by requesting a set of files as quickly as possible; it is necessary to provide some support for modeling the session-oriented nature of Web traffic. As a consequence, a benchmark that uses just a filelist is not able to reproduce a realistic Web workload. When using filelists, the only feasible alternative is to provide some support to define the characteristics of a user session (such as user think times) otherwise the workload generator will not be able to emulate a realistic load. Furthermore, the overall size of the file set being used should be checked to ensure that the server caching mechanism is fully exercised.

In the *analytical distribution-driven* approach, the Web workload characteristics are specified by means of mathematical distributions. The requests are issued according to the parameters of the workload model. The probability distributions may be used to generate random values that reproduce all the characteristics of the request stream during the execution of the benchmarking test. An alternative is to pre-generate all user sessions and the resulting sequence of requests, and to store them in a trace file which will be used by the workload generator. The analytical distribution-driven approach allows a tool to define a detailed Web workload characterization because all features are specified through mathemat-

ical models. Some can argue about the realism and accuracy of the workload characterization, but changing the parameters of a distribution or a distribution itself to evaluate the performance under different conditions is a really easy task.

The *hybrid* approach is a mix of the *filelist* and *analytical* techniques. For example, the objects to be accessed may be specified through a filelist, while session-oriented parameters, such as session lengths and user think times, are modeled through analytical distributions. In the hybrid method, parameters shaping the main characteristics of session-oriented workload are modeled through stochastic models.

Web Client Characterization. The first important characterization for a Web client regards the alternative between an open and a closed loop model. In a closed model, a pre-determined number of clients sends requests only after having received the previous server responses. Although this model does not give a realistic view of the offered load, it is adopted by several tools that aim to evaluate performance of a Web system subject to constant load. However, this behavior becomes unrealistic and not acceptable for a distributed Web-server system under heavy load conditions. Indeed, as Web traffic increases, clients spend most of their time waiting for responses and, substantially, they issue requests at the response rate imposed by the system responses. This situation is far from reality, in which the clients access a distributed popular Web site concurrently and independently from the server responses. Hence, an open client model, characterized by periodic client interarrival times, is typically preferred when evaluating the performance of a distributed Web-server system.

Another main feature related to the client requests is represented by the HTTP protocol that is supported by the emulated browser. The client should be capable of requesting objects using both HTTP/1.0 and HTTP/1.1. Indeed, the latter provides some interesting features (such as persistent connections, request pipelining, and chunked transfer encoding [20]) which affect the performance of the Web system under testing [8,19]. In particular, persistent connections are used to limit the number of opened TCP connections (thereby reducing resource consumption on the Web-server system) and to avoid slow start each time a new object is requested. It would be also important to have full support for various request methods (GET, POST, HEAD) in the request header. Further issues regard the possibility to allow for session tracking via cookies and to support SSL/TLS encryption in such a way to request secure Web services.

To properly mimic the resource usage of the Web-server system, the emulated client could also use multiple parallel connections for the retrieval of embedded objects in a Web page. Although this is a deprecated technique for its impact on the Web servers, it is commonly employed by modern browsers (together with closing active connection by means of TCP resets) to reduce the latency time experimented by users. This implies that the browser behavior cannot be naively emulated by a simple model in which the client opens a single TCP connection at a time for the retrieval of a single Web object.

4.2 Requirements for Distributed Web-Server Systems

In this section we identify the requirements pertaining to the workload characterization component which are suitable to perform the benchmarking of distributed Web-server systems. Besides the workload characteristics which should mimic at best those of real Web traffic and an open system model for client requests, the distinguishing feature that characterizes the benchmarking of distributed Web-server systems regards the mechanisms supported by the client for request routing.

No particular support is required to the benchmark of Web clusters, as the Web switch completely masks the distributed nature of the architecture to the clients that interact with the Web system as if it were a one server node. On the other hand, some request routing support must be provided for benchmarking geographically distributed Web-server systems in which multiple IP addresses may be visible to client applications. The most important feature to add to the client model is the DNS mechanism with all main steps related to the address lookup phase. This would allow us to test the impact of alternative routing mechanisms, such as DNS-based routing, URL rewriting, and HTTP redirection [13]. To support the last technique, the client should also be able to redirect the request as indicated in the response header.

4.3 Comparison of Selected Tools

In this subsection we analyze how the selected Web benchmarks specify their workload. We appreciate that most benchmark tools allow us to customize and extend the workload model in order to test different scenarios. On the other hand, the option for workload configuration of SPECweb and TCP-W benchmarks are quite limited because their goal is to measure the performance of different systems in a well-defined and standardized scenario. Obviously, we do not penalize these benchmarks for a limit that is intrinsic in their design.

Httperf permits two approaches to generate the request stream that is, hybrid and trace-based [32]. Both methods enable a session-oriented workload characterization and the requests for both static and dynamic services. In the hybrid approach, single or multiple URL sequences may be specified, together with some session oriented parameters, such as user think times. In the trace-based approach, user sessions are defined in a trace file. The requests are issued according to an open model. Both HTTP/1.0 and HTTP/1.1 protocols are fully supported, including cookies (although only one cookie per user session). Primary SSL support is provided, including the possibility of specifying session reuse, which is an important feature as it avoids handshaking every client request. Httperf allows also to specify some realistic browser characteristics, such as the use of multiple concurrent connections.

SURGE relies on a analytically generated workload aimed at dealing with the self-similarity issues of the Web characteristics [7,8]. The workload model derives from empirical analysis of Web server usage to mimic real-world traffic properties. In SURGE, the workload is measured in terms of *User Equivalent*,

defined as a single process in an endless loop, alternating between requests and thinking times. Therefore, the user behavior is modeled as a bursty two-state ON/OFF process, where ON periods correspond to the transfer of Web objects, and OFF periods correspond to the silent intervals after that all objects in a Web page have been retrieved. It has been demonstrated that the superposition of a large number of ON/OFF sources results in self-similar traffic, if the durations of ON and OFF phases are described by heavy-tailed distributions [12, 43]. The characteristics of the request stream are specified through heavy-tailed distributions as regarding file size, request size, file popularity, embedded object references, temporal locality, and OFF times. Support for HTTP/1.0 and HTTP/1.1 protocols is provided (the latter with request pipelining), while no security support is provided. The browser activity is emulated using only one connection at time. SURGE remains the most accurate tool for the characterization of static requests. Its main limits, especially for the analysis of multi-tier Web systems, are that the workload model does not take into account request for dynamic services and that the generation of requests follows a closed-loop model.

The S-Clients workload is intentionally not realistic, being characterized by a single file which is requested at a specified fixed rate [6]. This choice provides excellent measurements of the server performance and capacity, but does not exercise other system resources, starting from the disk as the file is always get from the cache. With S-Clients it is not possible to specify any browser behavior, only the plain HTTP/1.0 protocol is supported, and no session encryption is allowed. These aspects make the workload characterization provided by S-Clients inappropriate from the point of view of the workload realism, while it is appreciable its sustained load solution for stress testing distributed Web-server systems, as discussed in Sect. 6.

WebStone denotes the characteristics of the request stream through a file list [30]. The benchmark workload includes both static and dynamic services, the latter generated through CGIs and server APIs. Since the maximum size of the filelist is limited to 100 files, it is difficult to model typical workloads of distributed Web-server systems which consist of thousands of files. Moreover, there is no way of specifying a session-oriented workload, since requests are intended to be issued consecutively. The workload is generated following a closed loop model. The emulation of the browser characteristics is quite limited, as WebStone supports only standard HTTP/1.0 without keep-alive. Support for encryption and authentication is not officially included, although a patched version exists which enables it [31].

WebBench follows a hybrid approach, where the workload characterization is done through test suites that is, appropriate combinations of request streams (which model specific user interactions) along with their reproduction modalities [45]. Static, dynamic (CGI and API), and secure services may be configured. Both HTTP/1.0 and HTTP/1.1 protocols are supported. The two main drawbacks are related to the impossibility to specify the session-oriented nature of client requests and to the closed loop model.

Web Polygraph permits a fairly complete specification of the Web workload, characterized by a session-oriented request stream, Web pages, popularity of files, cacheability at the client, server delays due to network congestion [42]. Many of these propierties may be specified through probability distributions. Requests may be issued through both HTTP/1.0 and HTTP/1.1 protocols, in an open or closed loop model. An interesting feature is the presence of already configured Web workloads oriented to layer-4 and layer-7 Web clusters.

A different common observation is in order about the workload of SPECweb99 and TPC-W benchmarks. They define standardized Web workloads which are not intended to be customized by the user. Hence, they cannot be used to define workloads for different categories of Web sites. The basic workload of SPECweb99 [38] includes both static and dynamically generated content, while an enhanced version supports also secure services [39]. The static workload is characterized by four classes of file sets, modeling different types of Web servers and spread into a precomputed number of directories. Directory access and class access are chosen according to a Zipf distribution. The dynamic workload models two common features of commercial Web servers: advertising and user registration. The client model is closed because a fixed number of clients is executed during each experiment.

The TPC-W benchmark specification (note that it is not a tool) defines the details of the Web services and content at the site and the workload offered by clients [26,41]. It specifies a database structure oriented to e-commerce transactions for an online bookstore together with its Web interface. Clients are characterized by Web interactions that is, well-defined sequences of Web page traversals which pursue particular actions such as browsing, searching, and ordering. Request streams are session-oriented, with think times between Web page retrievals. It also includes secure connections because some client actions (e.g., online buying) require SSL/TLS encryption.

No tool provides explicit support to DNS routing that is of key importance in geographically distributed Web-server systems and Content Delivery Networks. Most benchmarking tools perform only one DNS lookup at the beginning of the test, that is unrealistic since a DNS lookup is needed per each client session. There is no much support even to other (application based) request routing mechanisms, for example only Web Polygraph and WebBench support HTTP redirection.

5 Content Mapping on Servers

An interesting issue of a benchmarking tool for distributed Web-server systems is the replication of the synthetic content among the multiple server nodes. Once the synthetic workload has been specified, it must be replicated on the Web nodes composing the Web-server system prior that the workload generation engine starts to generate the request stream. This constitutes an error-prone operation which should be automated as much as possible. Another peculiarity of distributed Web-server systems is that the replication strategy may differ on

the basis of the system architectures, because the content is not always replicated among all the servers.

Let us first examine the problem of mapping the Web site content onto the Web servers in the case of one Web server, for which we identify three alternatives: *full support*, *partial support*, and *no support*.

The most attractive feature to the benchmark user is a full support that is, once the benchmark user provides the specification of the entire Web site content (the tree of static documents as well as the set of data to be placed on the back-end servers), it is automatically generated and uploaded on the Web and back-end server disks. A partial support means that only a portion of the Web site content (that , static documents) is put on the server disk, while other content (that is, dynamic services) is left up to the benchmark user. If the benchmark does not provide any support for the content generation and mapping, the content must be generated and uploaded manually on the server. Manual generation is errore-prone and is often unfeasible due to the large number of involved files. Thus, the presence of a mapping component is strongly encouraged.

Webstone provides a partial support for Web content creation [30]. It is possible to specify and generate a set of static files with given sizes, while dynamic content creation is left to the user. The other Web benchmarking tools, although providing in some cases already predefined Web contents (SPECweb99, WebBench), neither perform content mapping across different Web servers nor install them. Every decision is left to the benchmark user.

The benchmark study of a distributed Web-server system has an additional requirement because the site content may be fully replicated, partially replicated, or partitioned among the multiple server nodes. The two last configurations are typically used to increase the secondary storage scalability [10,44] or to enhance the features of specialized server nodes providing dynamically generated content or streaming media files. It is also important to observe that fully replication can be easily avoided only if we use a layer-7 Web switch that can take content-aware dispatching decisions. An alternative is to use a layer-4 Web switch combined with a distributed file system, because any selected server node should be able to respond to client requests for any part of the Web site content.

We can easily observe that none of the selected benchmarking tools includes any utility for fully or partial content replication among multiple servers.

6 Workload Generation Engine

An important component of a Web benchmarking tool is the workload generation engine, which is responsible for reproducing the specified workload in the most accurate and efficient way.

Distributed Web-server systems are characterized by a huge number of accesses, which have to be emulated with a usually limited amount of resources. This may be obtained by generating and sustaining overload [6] that is, constantly offering a load that exceeds the capacity of the distributed Web system. In this section, we identify the main features of workload generation, analyze the

requirements that are specific for distributed Web systems, and discuss how the selected Web benchmarking tools behave with respect to the identified features and requirements.

6.1 Classification of Alternatives

The two main features in a workload generator are the *engine architecture* denoting the computational units used to generate Web traffic (processes or threads) and mutual interactions, and the *coordination scheme* defining the ability of configuring and synchronizing the computational unit executions.

Engine Architectures. We give a possible taxonomy of workload generator architectures in Fig. 4. In a *centralized architecture*, a single instance of the workload generator runs on a single node, whereas in *distributed architectures* the engine is spread across multiple nodes.

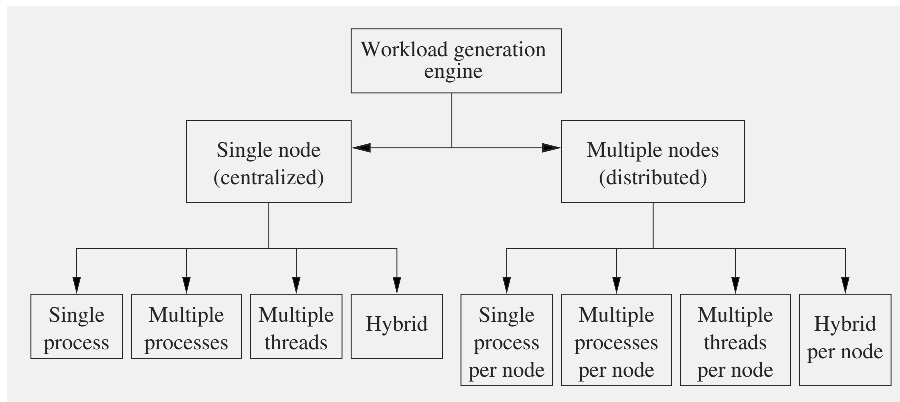

Fig. 4. Architecture of a workload generator.

The architecture characterization defines the nature of the computational units on each client node. In *single-process architectures*, one process is responsible for the generation of the whole workload on the node on which it is running. In *multiple-process architectures*, the task of generating client requests is split up among several user-level processes. The multiple-process approach is relatively straightforward, but suffers from two drawbacks. First, it is CPU-intensive because of frequent context switches, especially when many user processes are spawned on the same machine. Second, since process address spaces are usually separated, most information (e.g., the workload configuration) must be replicated, thus wasting main memory that is an important resource for the scalability of the load generated by the client node.

In *multi-threaded architectures*, light-weight processes sharing the same address space are used to generate the appropriate portion of workload, while in *hybrid architectures* each node runs several user processes, each handling multiple threads. The multi-threaded architecture does not suffer from context switch drawbacks. In general, light-weight processes guarantee for a better scalability, but multi-threaded programming incurs in a higher degree of complexity. Sharing the address space surely leads to a better memory utilization than in the multi-process architecture, at the cost of implementing synchronization primitives which could block client activity. Finally, several threads usually share one set of system resources, which could be exhausted (for example, the file descriptor set used to reference TCP connections).

The hybrid architecture aims to combine the advantages of multi-threaded architectures (lower CPU overhead due to less frequent context switches) with those of multi-process architectures (increase in available system resources such as socket descriptors).

Coordination Schemes. The task of configuring and coordinating the execution of the computational units may be performed manually or automatically. In the latter case, two coordination schemes are possible: *master-client* and *master-collector-client*.

In the master-client scheme (see Fig. 5), the client generation task is delegated to a *master component*, which reads the configuration and performs several operations. First, it decides how many computational units have to be started and how they are distributed among the client nodes, in order to offer the specified workload. Then, it distributes part of the workload specification (for example, the filelist) among all clients. Finally, it synchronizes the start of the benchmarking experiment.

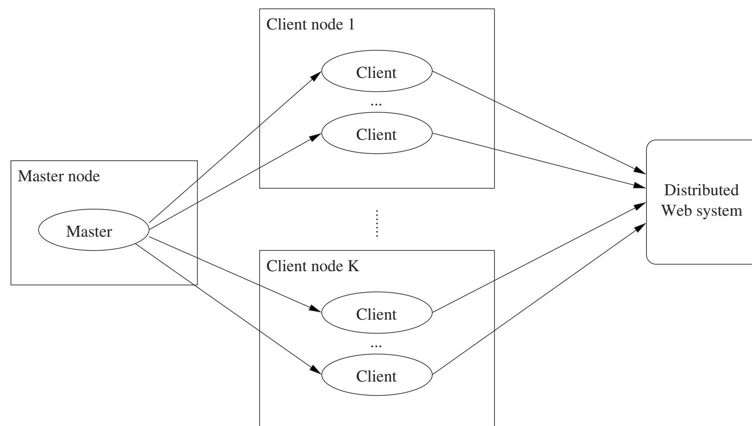

Fig. 5. The master-client coordination scheme.

The master-client approach is further extended in the *master-collector-client* coordination scheme, illustrated in Fig. 6. One or more collector processes are activated on each client node, either manually or automatically through a master process (for clarity of representation, Fig. 6 shows only one collector). The master connects to each collector, distributes the workload configuration, and synchronizes the start of the benchmarking experiment. Each collector reads its portion of configuration from the master, spawns the necessary amount of computational units, and waits for a start signal from the master. Master, collector, and clients are logically separated, but they may reside on the same node.

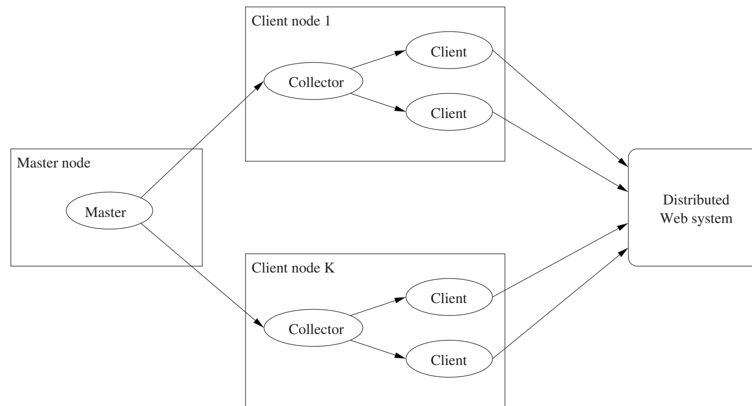

Fig. 6. The master-collector-client coordination scheme.

We can conclude that is clearly preferable to have an automated generation of client emulators among different nodes than referring to manual activations. This is especially true if the coordinator is able to share the Web workload among multiple nodes according to the capacity of each client node.

6.2 Requirements for Distributed Web-Server Systems

Distributed Web-server systems are typically subject to a large amount of traffic, which has to be reproduced somehow to evaluate their performance under realistic conditions. Thus, the *scalability* of the workload generation engine is a strict requirement for a benchmarking tool. Single node architecture is not adequate since operating system resource constraints typically limit the number of concurrent clients. The generation of Web workload should be distributed across as many nodes as possible. The amount of available client nodes for tests is usually limited to few tenths. Thus, it is desirable to generate the maximum amount of workload on a given node. This holds especially when client nodes have heterogeneous capacities. An unbalanced assignment of workload may determine the under-utilization of some nodes and the partial inability to generate requests on others.

Being the scalability of the workload generation engine an important requirement for the performance evaluation of distributed Web systems, we observe that it is quite difficult to achieve it using a centralized architecture. The only solution to keep one process spawning many concurrent client sessions is to use an event-driven approach, combined with non-blocking I/O [18]. This single process polls the network for events and reacts accordingly. This approach involves programming non-blocking I/O, which can be tricky and much more difficult than in a multi-process or multi-threaded model. Moreover, one process may run out of file descriptors if the machine is not well tuned. On the other hand, this approach does not suffer from context switch overheads, provided that the client node does not execute other resource intensive tasks.

6.3 Comparison of Selected Tools

In this section we analyze how the selected benchmarking tools generate the load offered to the Web system.

Httperf generates the specified workload through one process, implementing an event-driven approach with non-blocking I/O [32]. As a consequence, the workload generator keeps a single CPU constantly occupied, so it is recommended not to run more than one httperf process per CPU. Furthermore, the maximum number of concurrent sessions is bounded by typical process limits such as the maximum number of open descriptors. As there is no coordination scheme, several instances of httperf must be executed manually on distinct nodes to scale to the desired workload; an helper utility can be used to automate this task [29]. The workload generation engine of httperf is adequate to the performance evaluation of distributed Web systems.

In SURGE the client activity is modeled through a User Equivalent, which is represented by a thread [7,8]. The benchmarking experiment is activated by invoking a master which spawns a predefined number of client processes. Each client process generates a prefixed number of client threads (i.e., User Equivalents). Therefore, SURGE architecture can be defined as being centralized, multiple-process and multiple-thread. The coordination scheme is a master-collector-client, although on a single node. Since no support is provided to automatically distribute clients among multiple nodes, several instances of the SURGE master have to be activated manually on distinct client nodes, in order to scale the workload.

The workload generator of S-Clients is executed by a single process on one client node [6]. The engine aims at generating excess load by using non-blocking connects and closing the socket if no connection was established within a given interval. There is no means to automatically start different workload generators on distinct nodes, but this operation has to be performed manually. Furthermore, since timers are implemented using the *rdtsc* primitive [14], the ability to generate connections with a specified rate depends heavily on the CPU speed of the client, and the CPU type, which should be a Pentium. The most interesting feature of S-Clients for the benchmarking of distributed Web systems is the use

of non-blocking connections combined with timeouts, as it allows to guarantee a specified connection rate.

In WebStone, the activity of Web users is emulated through a preconfigured, fixed number of *Web clients* [30]. The architecture of WebStone is distributed and multi-process; each Web client is executed as a distinct user process that requests files continuously. Web clients are distributed over several nodes through another user process, called *Web master*. The workload generator of WebStone is not able to sustain high loads and, consequently, it is not adequate for the performance evaluation of distributed Web-server systems.

In WebBench, Web clients are emulated through client processes running on distinct nodes [45]. The architecture of WebBench is distributed and may be either single-process or multi-threaded. In the first case, each client runs as a user process (called *physical client*), in the latter, multiple clients run as threads (called *logical clients*). A controller on a distinct node coordinates the client execution. The recommended coordination scheme is master-client with one physical client per node. It is also possible to specify a master-collector-client scheme where logical clients are locally coordinated. In both cases, processes residing on client nodes must be started manually. The features of the WebBench workload generation engine are not sufficient for the benchmarking of distributed Web-server systems.

The workload generation engine in Web Polygraph has a centralized, single-process architecture, which is capable of sustaining overload [42]. Optionally, server agents may be used to emulate parts of a distributed Web server system, besides exercising real components. A drawback of Web Polygraph is the lack of some support to automatically distribute the generation of requests across multiple client nodes.

The TPC-W benchmark specification requires that client requests be issued by a given number of "emulated browsers", which remains constant throughout the experiment [26,41]. The number of clients is obtained as a function of the database table size and appropriate scaling factors. As a consequence, it is difficult to generate a considerable amount of traffic without modifying the Web content.

SPECWeb99 distributes clients on several machines in order to achieve workload scalability [38]. If the operating system supports POSIX threads, clients are executed as threads, otherwise as processes. Thus, the architecture of the SPECWeb99 engine is distributed and multi-process or multi-threaded. Clients are executed by processes called *collectors*, which must be manually activated before starting the test. A master process connects to the collectors, sends them the configuration parameters and synchronizes the runs. The workload generation allows for a certain degree of scalability but cannot be sustained when the distributed Web system is under stress.

7 Data Collection and Analysis

The measurement and collection of data during the benchmarking test is of key importance. If done superficially, it leads to improper conclusions about the performance of the resources constituting the system. A first issue concerns the definition of the metrics and the statistics which can yield the most useful information about the components of the distributed Web-server system. Then it is important to investigate the data collection strategies that are somehow related to the previous choices.

7.1 Classification of Alternatives

The most common metrics for Web system performance are reported in Table 2 [28].

Table 2. Typical Web performance metrics.

Name	*Meaning*
Throughput	The rate at which data is sent through the network
Connection rate	The number of open connections per second
Request rate	The number of client requests per second
Reply rate	The number of server responses per second
Error rate	The percentage of errors of a given type
DNS lookup time	The time to translate the hostname into the IP address
Connect time	The time interval between the initial SYN and the final ACK sent by the client to establish the TCP connection
Latency time	The time interval between the sending of the last byte of a client request and the receipt of the first byte of the corresponding response
Transfer time	The time interval between the receipt of the first response byte and the last response byte
Web object response time	The sum of latency time and transfer time
Web page response time	The sum of Web object response times pertaining to a single Web page, plus the connect time
Session time	The sum of all Web page response times and think times in a user session

As Web workload is characterized by heavy-tailed distributions, most performance metrics may assume highly variable values with non negligible probability. Collecting just minimum, mean, and maximum times, error levels, is not an error, but these metrics may not yield a representative view of the system behavior. The metrics subject to high variability should be represented by means of higher moments, percentiles or cumulative distributions [16,22]. Mean values may be

meaningless about peaks due to heavy load. This holds for throughput and response times (specifically, object and page response times), which may exhibit high variations from the mean value.

These performance statistics require more expensive or more sophisticated data collection strategies, because measurements should be collected and stored to allow later creation of histograms. The alternative is to implement techniques to dynamically calculate the median and other percentiles without storing all observations [17]. Let us analyze the main approaches to the *collection strategy* (that is, *record storage*, *data set processing*, and *hybrid*) and *output analysis* that are strictly related.

In the *record storage* approach every record is stored. The generation of meaningful statistics is entirely delegated to the output analysis. This technique allows us to easily compute histograms and percentiles but it requires enormous amount of memory. The main memory is often not sufficient, and the use of secondary memory introduces other problems, such as delays and possible interferences in the experiment. Moreover, the elaboration of great amounts of data tends to be resource expensive even if done post-mortem. Actually, a complete collection and processing of all measurements is seldom necessary, and the use of sampling techniques is the best alternative when we want to use the record storage approach.

In the *data set processing* approach, measurements are not stored directly into some repository, but are used to keep updated the *data set* with the interesting statistics. Data set processing does not use great amounts of system resources such as CPU or memory. This is the standard way for computing performance indexes which do not require sophisticated statistics, such as minimum, maximum, and mean values. It would be also possible to implement techniques that dynamically calculate the median and other percentiles without storing all observations [17], but even these more complex computation may interfere with the experiment. The data set may coincide or not with the set of parameters presented as final statistics. When they do not coincide, the generation of useful statistics is partially delegated to the output analysis component that processes the data set at the end of the benchmarking test.

None of the previous techniques is clearly the best. However, we can observe that sophisticated statistics are really necessary only for those metrics which are subject to high variance. In many other cases, min, max and mean values are acceptable. For this reason, we consider also the *hybrid* approach that is a mix of the previous two techniques. Each measurement may be stored, processed to keep updated a data set, or both. This approach leads to a better trade-off between main memory resource utilization and usefulness of the collected data. The performance indexes that do not require sophisticated statistics may be computed at run time, for the other indexes we can store the relative measurements and postpone the evaluation during the output analysis after the experiment.

When multiple client emulators are used, it is necessary to use the data sets and samples stored by each of them to compute the final metrics which are

presented to the benchmark user in a clear form. This operation is mandatory in the case of distributed Web-server systems.

7.2 Requirement for Distributed Web-Server Systems

A typical benchmarking tool for distributed Web-server systems distributes the generation of high volumes of Web traffic across different client nodes. Data collection is usually done at the level of each computational unit. While it is good to have per-process (or per-thread) statistics, it is certainly crucial to have global reports, to understand how well the whole system has performed. To obtain global session statistics, cumulative distributions and percentiles (not only per-node statistics), data sets and records must be aggregated before the computation of global statistics. Therefore, *aggregation of collected data* is a key feature that should characterize all tools for distributed Web-server systems. We also consider important to have *session-oriented statistics* that is, final reports including metrics relative to user sessions, in addition to global statistics, which are quite useful for evaluating the performance of the whole system.

Besides the previous considerations, there is a serious problem that makes traditional benchmarking tools for Web servers not useful to collect important statistics for an accurate performance evaluation of distributed Web-server systems, especially in the design and prototype phase when different alternative architectures and solutions must be evaluated. Indeed, all considered tools have been designed for the interaction of multiple clients with one server and give global metrics that cannot take into account that the server side consists of multiple components usually running on different machines. In a distributed Web-server system, the interaction of the client with this system consists of several steps, such as switching to the right server and invoking the appropriate process for the generation of dynamic content. The delay of each of these phases makes up for the response time seen by the clients. A high response time means a bad system performance, but it does not indicate where the bottleneck is. The associated overhead within each phase of the Web transaction must be measured and evaluated, since bottlenecks in one component make the whole system slower. Some of the phases of a Web transaction in a distributed system are very hard (if not impossible) to measure at run time without making modifications to the system components. For example, in a locally distributed Web system, the time required by the switch to dispatch a client request cannot be measured from the client side. In other cases, the performance of some components may be inferred by the external performance metrics. For example, in one-way Web clusters with a layer-4 Web switch, the initial client SYN is processed by the switch and sent to the appropriate Web server, which establishes the TCP connection. Thus, the *connect time* embodies switch and server latencies, leaving us with the doubt about a potentially overloaded node. Instead, the *latency time* is an approximate measure of server performance, since TCP segments do not pass by the switch once the connection has established with the appropriate server.

In layer-7 one-way Web clusters, the opposite is true. *Connect time* is an approximate measure of switch overload, since it establishes TCP connections

with clients prior to assigning requests to the appropriate servers. On the other hand, the *latency time* embodies the Web switch and server delays, since every client TCP segment directed to a server passes through the Web switch. In this case, the latency time does not give sufficient information to localize a possibly overloaded Web system component.

If one-way architectures allow an approximate evaluation of component performance, this estimation is practically impossible in two-way architectures, since both packet flows pass through the switch. Hence, the above mentioned procedure may lead to gross evaluation errors. In general, there is no way for measuring the performance of the Web switch and the single servers through client measurements. Therefore, the right approach is that of enabling logging at every system component and analyzing the resulting logs at the end of the test. Monitor facilities and a *log analyzer* are required to this purpose. They should be highly configurable because different applications may have different logfile formats. Analyzing log outputs may require integration or modifications of the network application software because the standard logs have too coarse granularities (e.g., 1 second in the Apache server). Moreover, the statistics obtained by the internal monitors must be integrated with those of the benchmark reports.

For geographically distributed Web systems, it is necessary to measure the time taken by the request routing mechanism, such as DNS lookup and request redirection times.

7.3 Comparison of Selected Tools

Httperf collects a large variety of metrics, both session- and request-oriented [32]. The most interesting non-session oriented metrics are connect time, latency time, request and reply rate, throughput and error rates. Response time at the granularity of Web objects is not collected; session-oriented metrics include session time and session rate. For each of these metrics, minimum, mean, maximum values and their standard deviations are computed through data sets. Support for record storage through histograms is given only for some metrics such as session length and connection duration. Httperf has a hybrid data collector and a centralized output analyzer. It also performs hybrid processing of the collected data. A final report is presented with global and per-session statistics, thus providing a way for detecting the degree of user concurrency in a distributed Web-server system. However, it requires some extensions. For example, it would be interesting to have the Web page response time as a metric. Moreover, the records should be stored in histograms for later processing to evaluate higher order statistics.

SURGE stores only records of transaction time and Web object size for later processing [7,8]. The output analyzer of SURGE is centralized and oriented to record processing. It operates on server logs (in common log format) and on the log file generated at the end of the benchmarking experiment. Final metrics provided to the benchmark use are session-oriented; a log-log cumulative distribution table of Web page response times is also provided.

S-Clients collects a data set consisting of connection life time sums (which are used to approximate transaction times, since HTTP/1.0 is used) and global

counters of opened connections and successfully delivered responses [6]. S-Clients presents only request rate and average response time of the requested URLs.

WebBench keeps at run-time the following data set: a global count of successful requests, a sum of transaction times and a sum of transfer sizes [45]. These data sets are required to compute the final metrics, that is, number of requests per second and throughput. WebBench gives only two overall metrics: interaction times per second and throughput in bytes per second. They are computed locally on each client and centrally gathered by the controller.

Webstone uses a hybrid data collector and output analyzer [30]. The retrieval phases of a Web object (connect latency and transfer times) are marked by timestamps which are all recorded, while global counters are kept through appropriate data sets. WebStone provides a report with global and per Web object connect times, response times, error rates. It also computes a global connection rate and a metric known as *Little's Load Factor* [28]. No session oriented metrics are reported, no response time subdivision in latency and transfer is evaluated, although the collected records allow a successive computation.

The data collector of Web Polygraph is hybrid [42]. It stores records for later computation of reply size and hit/miss response times. It also keeps global counts for error rates, client and server side throughput, cache hit ratio and byte hit ratio. No provision for session oriented metrics is provided. In Web Polygraph, each client and server agent process generates its own log file. They have to be manually concatenated before processing by the report module. Reports include several performance graphs for throughput, cache hits and misses response times, persistent connection usage, error rates.

SPECweb99 uses its own performance metric [38]. Substantially, it is the maximum number of connections supported by the Web server under certain conditions (throughput ranging between 300000 and 400000 bits per second). To this purpose, SPECweb99 collects throughput, request and response times over a single connection. According to online documentation [38], this is done in a data set way. There is no session oriented statistics. In SPECweb99, the output analyzer is centralized: data collected from each client is gathered by the master process which reports test results for that iteration. A report consists of summary, results, overall metric, and configuration information. The SPECweb99 metric is the median of the connection average result over 3 iterations.

The TPC-W benchmark specification defines the collection of the *Web Interaction response time* (WIRT), which is the time interval occurring between the sending of the first byte of a client request that starts a Web interaction and the retrieval of the last byte in the last response of the same Web interaction [26,41]. This is necessary to compute the final metric that is, the throughput of Web interactions per second. The specification also suggests running performance monitors on the servers for monitoring CPU utilization, memory utilization, page/swap activity, database I/O activity, system I/O activity and Web server statistics. The TPC-W benchmark specification defines three performance indexes: WIPS, WIPSb, WIPSo, that are counted as the number of Web interactions per second during shopping, browsing and ordering sessions, respectively.

The TPC-W specification recommends a report including graphs for the following metrics: CPU utilization, memory utilization, page/swap activity, system activity, Web server statistics (number of requests and error rates per second). No session-oriented statistics are planned, but the provided graphs should give an idea about the load conditions of the system.

8 Benchmark Support for Wide Area Networks

Benchmarking experiments of Web-server systems are usually carried out in a closed, isolated, and high-speed local-area network (LAN) environment. These laboratory experiments do not take into consideration network-related factors, such as high and variable delays, transmission errors, packet losses, and network connection limitations [28]. Modeling interactions that occur in the real Web environment using clients machines connected to the Web-server system by a low round-trip time, high-speed LAN may lead to incorrect results, because the provision of Web-based services in the real world involves wide-area network connections in which the presence of network components (such as routers) make the environment noisy and error-prone and have influence on Web server performance [33]. Therefore, to model interactions that occur in the real Web environment using both clients machines connected to the Web-server system by a low round-trip time, high-speed LAN may lead to incorrect results. Indeed, as a result of benchmarking experiments carried out in LAN environments, it occurs that performance aspects of the Web-server system that depend on the network characteristics are not exposed or inaccurately evaluated. As a consequence of WAN delays, Web server resources (such as listen socket's SYN-RCVD queue) remain tied up to clients for much longer periods and therefore the system throughput decreases. Furthermore, in the wide-area Internet, packets are lost or corrupted; this causes performance degradation as these packets have to be retransmitted.

To take into account WAN effect in the benchmarking of distributed Web-server systems two approaches are possible that is, *WAN emulation in a LAN environment* and *WAN environment*. The first consists in emulating the WAN characteristics in a controlled environment, where clients and server machines are interconnected through a LAN network, by incorporating factors such as delays and packet losses into the benchmarking tool. The WAN emulation approach allows to perform the tests in a controllable, configurable, and reproducible environment, allowing easy changes in test conditions and iterative analysis [6,33,37]. However, incorporating delays and packet losses due to WANs is not a trivial task. On the other hand, experiments performed in a WAN environment allow to identify many problems and causes of delays in Web transfers that do not manifest themselves in a LAN environment [9,5,21]. At the same time, these wide-area benchmarking experiments are hard to reproduce due to the uniqueness of the test environment.

In the WAN environment, the benchmarking experiments are carried out spreading the client machines in a wide area network. This approach suffers

from the difficulty in changing the network parameters of interest for different test scenarios. Furthermore, it may be hard to generate a high workload using it as discussed in [9], in which SURGE clients have been spread among different network locations.

The majority of currently available Web benchmarking tools that operate in high-speed LAN environment ignore the emulation of WAN conditions. Some efforts in this direction have been pursued in some already considered benchmarking tools (S-Client [6], WebPolygraph [42], and SpecWeb99 [38], although quite limited in the latter) and also in WASPclient [33].

There are two main approaches that aim to emulate WAN conditions in a LAN environment that is, *centralized* and *distributed*. In the centralized approach, one machine acting as a WAN emulator is interposed between the client machines and the Web-server system to model WAN delays and packet losses by dropping and delaying packets. S-Clients follows this approach, by putting a router between the S-Client machines and the server system aimed at introducing an artificial delay and dropping packets at a controlled rate [6].

In the distributed approach, each client acts as a WAN emulator, by directly delaying and dropping packets. WASPclient implements an interesting distributed approach [33], by using an extended Dummynet layer in the protocol stack of the client machines to drop and delay packets [36]. The centralized approach is transparent to the operating system of both client and server machines; however its scalability is limited [33]. On the contrary, the distributed approach has the advantage that it provides a higher scalability, but it requires modifications to the operating system of the client machines.

9 Conclusions

This study leads us to conclude that many Web benchmark tools work fine when used to analyze a single server system, but none of them is able to address all issues related to the analysis of distributed Web-server systems. Many popular tools, such as SURGE and Webstone, suffer age problems, as they do not support dynamic requests and more recent protocols. Very few of them consider application-level routing of the requests, such as DNS and HTTP redirection, URL rewriting. In summary, we notice the lack of ability to sustain realistic Web traffic under critical load conditions, the difficulty or impossibility of emulating realistic dynamic and secure Web services, the poor support in analyzing collected statistics different from min, max, mean values. Hence, we can conclude that there is a lot of room for further research and implementation in this area.

References

[1] M. Arlitt. Characterizing Web user sessions. *ACM Performance Evaluation Review*, 28(2):50–63, Sept. 2000.
[2] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large Web-based shopping system. *ACM Trans. on Internet Technology*, 1(1):44–69, Sept. 2001.

[3] M. F. Arlitt and T. Jin. A workload characterization study of the 1998 World Cup Web site. *IEEE Network*, 14(3):30–37, May/June 2000.
[4] M. F. Arlitt and C. L. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Trans. on Networking*, 5(5):631–645, Oct. 1997.
[5] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz. TCP behavior of a busy Internet server: Analysis and improvements. In *Proc. of IEEE Infocom 1998*, pages 252–262, San Francisco, CA, Mar. 1998.
[6] G. Banga and P. Druschel. Measuring the capacity of a Web server under realistic loads. *World Wide Web*, 2(1-2):69–89, May 1999.
[7] P. Barford and M. E. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proc. of ACM Performance 1998/Sigmetrics 1998*, pages 151–160, Madison, WI, July 1998.
[8] P. Barford and M. E. Crovella. A performance evaluation of Hyper Text Transfer Protocols. In *Proc. of ACM Sigmetrics 1999*, pages 188–197, Atlanta, May 1999.
[9] P. Barford and M. E. Crovella. Critical path analysis of TCP transactions. *IEEE/ACM Trans. on Networking*, 9(3):238–248, June 2001.
[10] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed Web-server systems. *ACM Computing Surveys*, 34(2):263–311, June 2002.
[11] V. Cardellini, M. Colajanni, and P. S. Yu. Geographic load balancing for scalable distributed Web systems. In *Proc. of IEEE MASCOTS 2000*, pages 20–27, San Francisco, CA, Aug./Sept. 2000.
[12] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Trans. on Networking*, 5(6):835–846, Dec. 1997.
[13] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2616, June 1999.
[14] Intel Corp. Using the RDTSC instruction for performance monitoring, July 1998. `http://cedar.intel.com/software/idap/media/pdf/rdtscpm1.pdf`.
[15] A. K. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large-scale Web server access patterns and performance. *World Wide Web*, 2(1-2):85–100, Mar. 1999.
[16] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, 1991.
[17] R. Jain and I. Chlamtac. The P-Square algorithm for dynamic calculation of percentiles and histograms without storing observations. *ACM Communications*, 28(10), Oct. 1985.
[18] D. Kegel. The C10K problem, 2002. `http://www.kegel.com/c10k.html`.
[19] B. Krishnamurthy, J. C. Mogul, and D. M. Kristol. Key differences between HTTP/1.0 and HTTP/1.1. *Computer Networks*, 31(11-16):1737–1751, 1999.
[20] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement.* Addison-Wesley, Reading, MA, 2001.
[21] B. Krishnamurthy and C. E. Wills. Analyzing factors that influence end-to-end Web performance. *Computer Networks*, 33(1-6):17–32, 2000.
[22] D. Krishnamurthy and J. Rolia. Predicting the QoS of an electronic commerce server: Those mean percentiles. In *Proc. of Workshop on Internet Server Performance*, Madison, WI, June 1998.

[23] B. Lavoie and H. F. Frystyk. *Web Characterization Terminology & Definitions Sheet*. W3C Working Draft, May 1999.

[24] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. Traffic model and performance evaluation of Web servers. *Performance Evaluation*, 46(2-3):77–100, Oct. 2001.

[25] S. Manley, M. Seltzer, and M. Courage. A self-scaling and self-configuring benchmark for Web servers. In *Proc. of ACM Sigmetrics 1998 Conf.*, pages 170–171, Madison, WI, June 1998.

[26] D. A. Menascé. TPC-W: A benchmark for e-commerce. *IEEE Internet Computing*, 6(3):83–87, May/June 2002.

[27] D. A. Menascé and V. A. F. Almeida. *Scaling for E-business. Technologies, Models, Performance and Capacity planning*. Prentice Hall, Upper Saddle River, NJ, 2000.

[28] D. A. Menascé and V. A. F. Almeida. *Capacity Planning for Web Services. Metrics, Models, and Methods*. Prentice Hall, Upper Saddle River, NJ, 2002.

[29] J. Midgley. Autobench, 2002. `http://http://www.xenoclast.org/autobench/`.

[30] Mindcraft. WebStone. `http://www.mindcraft.com/webstone/`.

[31] N. Modadugu. WebStone SSL. `http://crypto.stanford.edu/~nagendra/projects/WebStone/`.

[32] D. Mosberger and T. Jin. httperf — A tool for measuring Web server performance. *ACM Performance Evaluation Review*, 26(3):31–37, Dec. 1998.

[33] E. M. Nahum, M. Rosu, S. Seshan, and J. Almeida. The effects of wide-area conditions on WWW server performance. In *Proc. of ACM Sigmetrics 2001*, pages 257–267, Cambridge, MA, June 2001.

[34] Neal Nelson. Web Server Benchmark. `http://www.nna.com/`.

[35] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison Wesley, 2002.

[36] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, Jan. 1997.

[37] R. Simmonds, C. Williamson, M. Arlitt, R. Bradford, and B. Unger. A case study of Web server benchmarking using parallel WAN emulation. In *Proc. of IFIP Int'l Symposium Performance 2002*, Roma, Italy, Sept. 2002.

[38] Standard Performance Evaluation Corp. SPECweb99. `http://www.spec.org/osg/web99/`.

[39] Standard Performance Evaluation Corp. SPECweb99_SSL. `http://www.spec.org/osg/web99ssl/`.

[40] Technovations. Websizr. `http://www.technovations.com/websizr.htm`.

[41] Transaction Processing Performance Council. TPC-W. `http://www.tpc.org/tpcw/`.

[42] Web Polygraph. `http://www.web-polygraph.org/`.

[43] 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. *IEEE/ACM Trans. on Networking*, 5(1):71–86, Jan. 1997.

[44] C.-S. Yang and M.-Y. Luo. A content placement and management system for distributed Web-server systems. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, pages 691–698, Taipei, Taiwan, Apr. 2000.

[45] Ziff Davis Media. WebBench. `http://www.etestinglabs.com/benchmarks/webbench/webbench.asp`.