A SIMULATION FRAMEWORK FOR CLUSTER-BASED WEB SERVICES

EMILIANO CASALICCHIO
Dipartimento di Informatica, Sistemi e Produzione
Università di Roma "Tor Vergata"
E-mail: casalicchio@uniroma2.it

RICCARDO LANCELLOTTI
Dipartimento di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
E-mail:riccardo.lancellotti@unimore.it

MARCO EMILIO POLEGGI
CERN-IT/INFN-CNAF
E-mail: Marco.Emilio.Poleggi@cern.ch

Abstract: We propose a simulation framework, namely CWebSim, specifically designed for the performance evaluation and capacity planning of cluster-based Web services. A broad variety of Web cluster configurations can be simulated through CWebSim. Its modularity permits the definition of different mechanisms, algorithms, network topologies and hardware resources. Also, two workload input alternatives are possible: a trace-driven mode and a distribution-driven mode that encompasses the most recent results on Web workload characterization. We present two case studies to show how CWebSim can be used to test cache cooperation protocols and Web switch dispatching algorithms.

Keywords: Simulation framework, Cluster-based Web systems, Performance evaluation, Caching, Cooperation algorithms

1. INTRODUCTION

Simulation is a common practice for the performance evaluation and capacity planning of Web-based systems. Indeed, the complexity of current Web architectures often makes analytical solutions of the related mathematical models infeasible. In this paper we present *CWebSim*: a simulation framework conceived for cluster-based Web services. These architectures are frequently used in practice: they are built on pools of server nodes, also known as *Web farms/clusters*, that are interconnected by a LAN with the goal of sharing the load of incoming requests. Many alternatives exist and CWebSim can be used to evaluate most of them, especially those acting at the higher levels, that is, application protocols, server-level caching, file systems. Nevertheless, CWebSim remains a detailed simulation model of a Web cluster, encompassing the main issues about the hardware, the operating system and the application layers, such as internal network and disk transfers, overheads due to request dispatching and processing. Special attention has been posed also to the workload model that reproduces a Web environment: in the case of the synthetic workload, realistic distributions for document sizes and requests are adopted, whereas the trace-based method has the appreciable feature of preserving the time dependencies. The CWebSim simulation framework can be customized into several Web service architectures, thanks to its modular design which allows the combinations of a broad variety of technical features, such as the adoption of different request dispatching policies and internal network hardware. By redefining the node functions and interconnections, complex proxy-caching systems or multi-tier architectures for e-commerce services can be easily simulated.

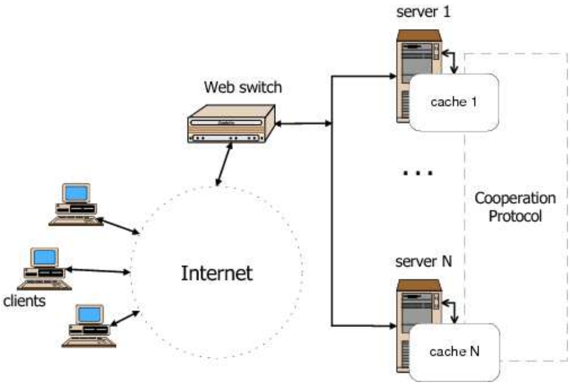


Figure 1: A basic cluster of cooperating Web servers

To the best of our knowledge, no simulation tool in literature is specifically oriented to Web clusters. General purpose frameworks exist for simulating computer networks, such as *ns-2*, *OPNET Modeler* and other tools that are considered in Section 5. CWebSim is written in C and uses the CSIM process-based simulation library (see Mesquite, 2001): this provides us with an adequate basis of classes and functions to be used as the building blocks for the implementation of complex simulation models. CWebSim can be ported to most operating systems thanks to the different CSIM distributions available: we tested it successfully on Linux and on various Unix-like platform. Some efforts can be necessary to implement CWebSim through other simulation languages/libraries, but there is no theoretical limit to its porting, because the design of CWebSim relies on CSIM features that are common to many other library-based simulation tools.

We present two case studies where we simulate through CWebSim a set of clustered HTTP servers. In the first case, the nodes cooperate for *global caching* purposes, as shown in figure 1, with the aim of improving the performance of standard Web clusters, composed of stand-alone nodes. The second case study focuses on dispatching algorithm alternatives that can be adopted at the front-end component of the Web cluster, namely, the *Web switch*.

The rest of the paper is organized as follows. Section 2 gives a detailed description of the simulation framework we designed for the evaluation of generic Web-based services. In Sections 3 and 4, we discuss the use of CWebSim for simulating, respectively, global caching mechanisms and Web switch dispatching algorithms. Some related work is discussed in Section 5. We outline conclusions and future work in Section 6.

2. CWebSim: A WEB CLUSTER SIMULATION TOOL

In this paper we present the architecture details and some applications of a simulation tool conceived for the performance evaluation of cluster-based Web architectures. *CWebSim* (‘C’ stands for “cluster”) is a discrete-event simulator implemented through the CSIM package: a library of routines for process-oriented simulations. The simulation framework underlying CWebSim can be customized to represent many classes of Web architectures, but in this paper we focus on locally distributed HTTP servers, also known as Web clusters. Therefore, in the following discussion we consider only the main components of Web clusters, that are the Web switch, the servers and the internal network, disregarding some external issues such as DNS servers, gateways and routers.

Figure 2 shows an high-level view of the CWebSim software architecture. CWebSim has a modular software structure conceived to isolate the implementation of the target model from the auxiliary simulation routines. The target system’s behavior is defined by a set of four modules (*Dispatching module*, *Client module*, *Server module*, and the subset of *Hardware definition modules*) that implement the core Web component models; these modules are described in Section 2.1. The remaining modules (*Input module*, *Output module* and *Gather module*) are described in Section 2.2: they implement simulation services such as input/output and statistic gathering routines.

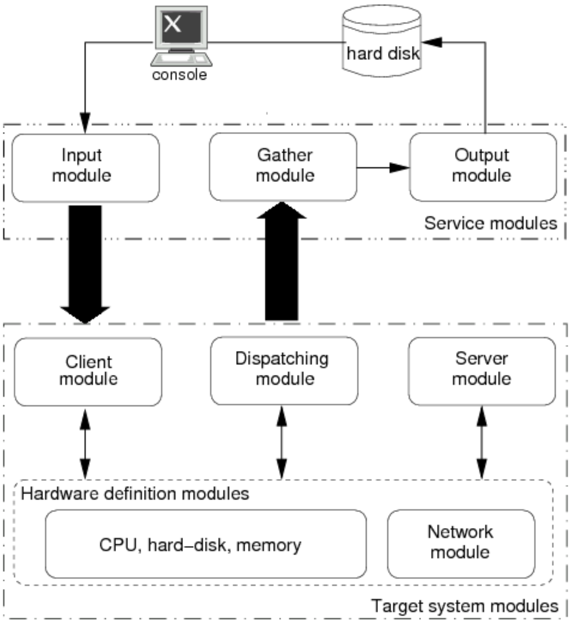


Figure 2: Software modules of CwebSim

2.1. Target system modules

The target system can be seen as a set of nodes that are interconnected through one or more network links. Each node is an abstraction of a physical computer unit, such as a PC or a workstation, and can be configured with different hardware capabilities, so that specialized nodes can be easily modeled. A process abstraction mechanism allows CSIM threads to be activated on the nodes. The main hardware components we consider are CPUs, hard-disks, memory banks and network interface cards (NICs): the related models are implemented by appropriate CSIM facilities having, if needed, their own queueing system. These components are included in the *Hardware definition modules*:

CPUs are round-robin-scheduled service centers. CSIM threads engage the centers for a *timeslice* configurable to approximate the behavior of current operating system schedulers. The service time depends on the requested operation.

Hard-disks are FCFS (First-Come-First-Served) centers with service time defined by a constant part (average values from off-the-shelf device datasheets are considered for the *controller delay* and the *seek time*) plus a part that is proportional to the requested data amount through a *transfer-rate* parameter.

Memory banks are not modeled as independent service centers: they are accessed through the CPU, like in real systems. The service time is proportional to the amount of the transferred data.

NICs can be defined according to various models available in the *Network module*, as described in Section 2.1.4.

As the main focus of the simulation is on Web-based applications, we find unnecessary to model low-level factors, such as operating system delays or MAC contention. Being typically two or three orders of magnitude lower than application service times, these overheads are negligible with respect to the costs of Web-based services hosted on a cluster.

A Web cluster model is obtained by defining a set of *Web server* nodes, a *Web switch* node and one or more *Web client* nodes. The "Web" qualifier in front of some non-ambiguous terms, such as "client" and "server", will be often omitted. The server nodes and the Web switch are interconnected by an internal network, whereas the client nodes are connected to the Web switch through an external network simulating the Internet; the network models are managed by the *Network module*. Since CWebSim is cluster-oriented, the overall target system description is based on some global data structures that define the components available on each node: for example, number of CPUs in a node, type of NIC, and so on. When the simulator is initialized, for each node (or group of homogeneous nodes), a setup function instantiates the needed CSIM facilities,stating, for instance, how much memory it owns and which scheduling policy its CPU adopts.

In a typical Web interaction, a client sends a connection request to the Web switch, that selects a server node and forwards the request to it; the server, in its turn, processes the request and sends a reply to the client. Our framework relies on process-oriented simulation. Hence, any active entity, such as clients, servers and dispatchers, are instances of CSIM threads, which communicate through internal message passing routines: models for high-level network protocols can be easily built upon this basic communication system.

2.1.1 Client module

This module is responsible of generating the input workload for the target system. The life cycle of a Web client is modeled according to the most recent results on the Web load characterization (see Barford, 1999, Arlitt, 1997). In a real scenario, users visit a Web site for a time whose length depends on their personal profile and on the requested services; once completed the service request, they leave the Web site. Hence, we consider a Web interaction model wherein clients enter the system and populate it for a *Web session*. During a Web session a client generates a random number of requests for *Web pages*, each of them being composed of an HTML file and a random number of *embedded objects*. Once received the requested document with its embedded objects, the client reads it and issues a new page request after a random user *think time*, mimicking the human behavior.

Clients are implemented by CSIM processes which generate input requests through either a distribution-driven model or a trace-driven model.

Distribution-driven model. Client processes are generated concurrently. At any simulation instant, the system is populated by a random number of processes. In our implementation, each client process is activated at a simulation time t_i , which is a stochastic variable describing the client arrival time: the mean difference value $t_{i+1} - t_i$ (inter-arrival time) can be adjusted to obtain the desired incoming load pressure.

Once activated, a client process computes some session parameters: first, the number of HTML pages requested during the Web session, and then, for each page, the number of embedded objects; HTML pages and embedded objects are chosen according to a certain popularity distribution. After this set-up phase, the Web client enters the system, issues the first connection request to the Web switch, and stands waiting for a reply. When a response message from one server of the cluster is received, the Web client process is resumed: it can either submit a request for an embedded object of the same HTML page or, if all the embedded objects have been received, it can wait for a user think time T_{tt}, during which the user is supposed to read the obtained page. These actions are repeated until all the HTML pages composing the Web session are received, then the client process leaves the system. The Web interaction model also covers connection refusals, which occur when the cluster service capacity reaches a predefined saturation point; rejected connections are not reissued, to avoid driving the system into a trashing state. Each random variable describing the client life cycle is characterized by a probability distribution function, whose shape and parameters can be defined by the CWebSim user. The alternatives of statistical distribution supported by CWebSim for distribution-driven workload generation are discussed in Section 2.2.1.

Trace-driven Model. In this model, all the characteristics of the workload model are determined by a real log of Web requests. The behavior of a client process is entirely driven by pre-loaded data. Our trace-driven model is fairly realistic because it preserves the time patterns of real logs of the typical Web traffic. We are mainly interested in preserving the time dependencies of the request stream, since this affects significantly the server performance. A trace log must be pre-processed to be used as input by the simulator: this operations introduces some artifacts necessary to rebuild a session-structured trace. The log's lines are scanned with a sliding time window that defines the maximum session time length: all requests coming from the same IP address within the time window are assigned to the same Web session. The first object requested in a session is considered an HTML page, whereas the following objects are treated as embedded objects. This introduces some approximations, but it is not possible to rebuild the exact page structure without considering the original structure of the Web site.

2.1.2. Dispatching module

The Web switch, also known as *dispatcher*, is responsible for forwarding the incoming Web requests to a server node selected according to a certain policy. CWebSim can simulate either stateless dispatching algorithms, such as *random* and *round-robin*, or stateful algorithms, such as *least-loaded* and *dynamically-weighted round-robin*. Also *content-aware* dispatching policies are supported. An overview of the main dispatching alternatives for Web clusters is given in Cardellini, 2002.

A Web switch can be a general purpose PC or a dedicated hardware device: in both cases the components relevant to our performance studies are CPU(s) and NICs. The basic Web switch node model consists of a queuing system with three service centers connected to work in a *one-way* mode, that is, only the incoming client requests go through it, whereas the server replies reach directly the clients. The service centers are a CPU used to run the dispatching algorithm, and two NICs: a first one connects the cluster to the Internet, through which the client requests come in, a second one connects the Web switch to the internal network which conveys the requests forwarded to the server nodes. A two-way Web switch could be modeled using two service centers for each NIC, according to the model of a Web server node proposed in Carrera, 2001.

Since some of the supported dispatching algorithms are based upon server state information, a special CSIM process runs on the Web switch node: every T_{get} seconds it stores load state information, such as number of active processes on each server node, server response time, CPU and disk utilization. The same information is used to simulate an optional admission control mechanism that rejects connection requests when the system gets overloaded.

2.1.3. Server module

The Web server node model encompasses the main hardware components, as shown in figure 3: a CPU, a hard-disk, a memory bank used as a main memory cache and two NICs. One NIC is used to connect the node to the internal network, the other NIC connects the node to the external network.

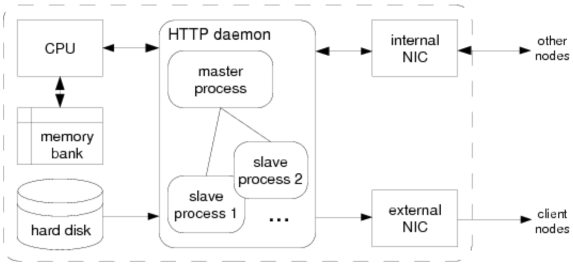


Figure 3: Server node model

We suppose that when a Web request is assigned to a Web server the service centers are visited as it follows.

1. The incoming HTTP request is queued at the **internal NIC** .
2. The **CPU** parses the incoming HTTP request, and runs a load management algorithm to decide whether to accept or discard the request.
3. The **memory bank** is accessed, engaging the CPU, trying to retrieve a cached copy of the requested document.
4. If the requested file is not found in the memory cache, the **hard-disk** is accessed to load the file into the cache.
5. The **CPU** is used again to produce an HTTP response.
6. The **external NIC** is used to deliver the response back to the client.

The Web server application simulates a multi-threaded server, whereby a CSIM process started at the system boot acts as a master HTTP daemon, waiting for request connections; when a server node receives an HTTP request from the Web switch, the master daemon forks a new slave CSIM process which serves the request. Any admission control policy is performed by the master daemon: the slave process is not spawned if the system is overloaded, in which case an error reply is sent to the client by the master process.

The main memory is used as a stand-alone local cache to simulate the behavior of the caching mechanisms of current operating systems. Different classical replacement policy, such as *Least Recently Used* (LRU), *Least Frequently Used* (LFU) and their variants, are supported.

We model both HTTP/1.0 and HTTP/1.1 protocols. Through the HTTP/1.0 protocol, a new slave HTTP process is forked to serve each Web page component, that is, the HTML file and all the embedded objects. Through the HTTP/1.1 protocol, the same slave HTTP process serves the HTML file and all embedded objects.

2.1.4. Network module

Since we are interested in the main issues of high-level communication protocols, such as the HTTP handshaking or the TCP connection handoff, the communication among the diverse Web entities is modeled as a single message exchange mechanism, without simulating any packet fragmentation and routing mechanisms. This approach aims to obtain a server-side performance evaluation, and is based on the assumption that each application-level computation, like the HTTP request processing inside the Web server process, is much longer than any characteristic time of the underlying network layers (with the exclusion of transfer delays). Inside a Web cluster, such a system configuration is achievable with the adoption of light-weight messaging protocols, like UDP. As for the geographical network overheads, the connection setup times are, in most cases, negligible with respect to the average transfer latency. Although for the server-side comparison of diverse cluster-based architecture it is not necessary to simulate a wide-area network, CWebSim can be easily extended to accommodate this need.

Various network models are implemented inside the Network Module of CWebSim. Each of them can be adopted according to the topology and features of the system under study.

Ideal network: neither delays nor contentions are considered. This model is suited to scenarios where all communication overheads are negligible, or where the network links should never become a performance bottleneck. NICs are dummy elements.

Delayed network: each transfer experiments a delay proportional to its size, but network contentions are not considered (NICs are delay elements). This model can be used when the network is supposed to never saturate, even though it is necessary to model transmission delays as in Internet’s back-bones.

Bus network: all the transfer requests share the same resource (a single FCFS queue service center, NICs are delay elements) and experience a delay proportional to the size of the transmitted data; network contentions are captured by the queuing discipline. This model is an approximation of Ethernet-like LANs, wherein each node is connected on the same physical medium.

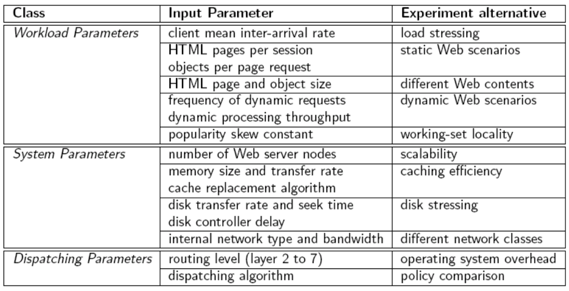
Switched network: it is composed of *N* independent links each of them being modeled through a single FCFS queue service center (that is the NIC), which simulates the network contentions; each transfer from one attached node to another engages two links with a delay proportional to the transmitted data size. Each link’s queue is used for bidirectional communication. No switching delay is considered, as, in most real cases, it is negligible with respect to the transfer latency. The maximum theoretical bandwidth of such network model is *N/2* times greater than the bandwidth value of the single link. This model resembles switched LANs with a star topology, like the Fast/Gigabit Ethernet.

2.2. Service modules

The service modules implement functionalities related to the simulator, such as providing the system with the input parameters, collecting simulation statistics and producing the simulation reports.

2.2.1. Input module

This module is responsible of processing all the input parameters, which are then dispatched to the other modules in order to configure and initialize their components. CWebSim’s input parameters can be divided into three main classes: *workload parameters* needed to configure the Client module, *system parameters* needed to configure the Web switch node, Web servers nodes and the Network module, and *dispatching parameters* which select and configure the dispatching algorithm to be used. Table 1 summarizes the main input parameters of CWebSim with some hints about the experiments that can be done by varying one or more of them.



Class	Input Parameter	Experiment alternative
Workload Parameters	client mean inter-arrival rate	load stressing
	HTML pages per session objects per page request	static Web scenarios
	HTML page and object size	different Web contents
	frequency of dynamic requests dynamic processing throughput	dynamic Web scenarios
	popularity skew constant	working-set locality
System Parameters	number of Web server nodes	scalability
	memory size and transfer rate cache replacement algorithm	caching efficiency
	disk transfer rate and seek time disk controller delay	disk stressing
	internal network type and bandwidth	different network classes
Dispatching Parameters	routing level (layer 2 to 7)	operating system overhead
	dispatching algorithm	policy comparison

Table 1: Input parameters of CWebSim

The workload parameters can be adjusted for different statistical distributions. Web access patterns exhibit a high variability and a self-similar nature which are well approximated by “heavy-tailed” distributions, such as Pareto and Lognormal distributions (see Arlitt, 2000, Barford, 1999, Cherkasova 2001). For instance, in Pitkow 1999 is shown that the number of requests per Web session follows an inverse Gaussian distribution, whereas a Pareto function fits the distributions of the number of embedded objects per request and of the user think time (see Barford, 1999, Pitkow, 1999). Since CSIM offers only a set of standard distributions, such as exponential and normal, we implemented inside CWebSim the following heavy-tailed distributions of interest for the Web: Inverse Gaussian, Lognormal, Pareto, Weibull and Zipf.

The case study presented in Section 3 shows some application examples.

2.2.2. Gather and Output module

After the definition of the system and of the workload model, the next step in the testbed setup is to choose the most appropriate *performance indexes* to evaluate cluster-based Web systems. We classify performance indexes into three broad groups: *service capacity indexes* , s*ervice efficiency indexes* and *system load indexes* . Any index can be referred either to the whole system or to its components.

Service capacity indexes estimate quantitatively the system performance at the service source point. Commonly adopted indexes are the following:

throughput: it is defined as the number of quantities computed by a system in a time unit. Quantities of interest for Web systems are: HTML pages (including the embedded objects), objects (files), HTTP and TCP connections, bytes. Some throughput indexes can be further specified: for instance, we can distinguish between the static object throughput for pre-existent files, and the dynamic object throughput for those files generated on the fly, like CGI results;

cache hit ratio: measurable as the ratio of the number of either documents (*document hit ratio*, or *DHR*) or bytes (*byte hit ratio*, or *BHR*) found in the cache to the total number of requested documents/bytes. The hit ratios give an estimate of the effectiveness of the caching subsystem, e.g., when simulating a proxy architecture.

The main service source points in a Web cluster are the Web switch (toward the server pool), each server node and the whole cluster (toward the client nodes).

Service efficiency indexes estimate *qualitatively* the system performance at the service destination point. The items of interest are:

response time: it is defined as the time experienced by an user to obtain a service from a system. In the case of Web systems, the service request can be an object/page request or a session, therefore, the response time is measured, respectively, as the time to receive an object, an HTML page with all its embedded objects or a certain number of HTML pages composing a session;

latency time: it is the time needed by a system to process a service request, excluding any communication delay. For instance, the object latency time of the entire Web cluster does not encompass the internal/external network delays.

The main service destination point in a Web cluster are the client nodes (the source being the whole cluster) and the server nodes (the source being the Web switch).

System load indexes estimate the stress of a system at the service processing point: they represent complementary performance indexes, that is, they give a measure of the service costs. We consider the following index:

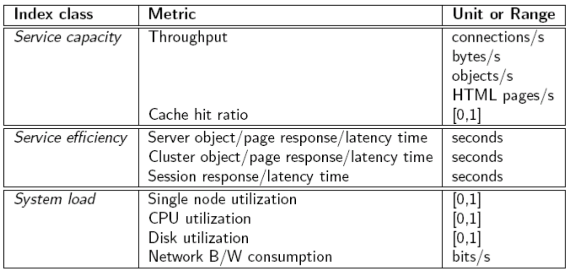
utilization: it is the fraction of a base time interval during which a single-resource system is busy. The utilization of a multi-resource system can be measured in many ways, according to its architecture. For an *n*−resource pipelined system, which is a good approximation of a server node equipped with one CPU and one disk, we adopt an OR-based definition of utilization: the fraction of a base time interval during which *at least one* of the system resources is busy. On the other hand, for a parallel system, like a dual CPU/disk server node, we adopt an AND-based definition of utilization: the fraction of a base time interval during which all the system resources are busy;

network B/W consumption: it measures the traffic conveyed by the network in a time unit, that is, the bandwidth (B/W) consumed: this gives an estimate of the network stress.

Once defined the performance indexes, we use some metrics to extrapolate a single or a set of values that are representative for that index. Common statistical measures of a set of sampled data are: *sample mean*, *sample standard deviation*, *x-percentile* and c*umulative distribution function*. Sample mean and standard deviation are representative measures of a set of sampled data following a standard distribution, such as normal and Poisson. When the probability distribution of the samples is characterized by an infinite standard deviation the most representative statistical metrics are the *x* -percentiles and the cumulative distribution functions. All these metrics can be easily extrapolated via CSIM routines.

In order to generate the output statistics, the Gather module provides the simulator with a service agent that, periodically, collects some status values probed inside the system components, computes some performance metrics and stores the resulting values into CSIM tables. The simulation can be controlled, through internal CSIM routines, in order to stop the run when a given confidence interval is reached. At the end of the simulation, the collected data are processed by the Output module to carry out a detailed simulation report. The main output statistics available in CWebSim are shown in Table 2.

Furthermore, the simulator is enriched by a set of Perl scripts that analyze the simulator output for additional computation. The scripts can build automatically graphs from a set of simulations or can aggregate multiple simulation runs (e.g., by calculating average values and standard deviation) to provide results that are more significant from a statistic point of view.



Index class	Metric	Unit or Range
Service capacity	Throughput	connections/s bytes/s objects/s HTML pages/s
	Cache hit ratio	[0,1]
Service efficiency	Server object/page response/latency time	seconds
	Cluster object/page response/latency time	seconds
	Session response/latency time	seconds
System load	Single node utilization	[0,1]
	CPU utilization	[0,1]
	Disk utilization	[0,1]
	Network B/W consumption	bits/s

Table 2: Output statistics of CWebSim

3. SIMULATION OF GLOBAL CACHING MECHANISMS

A classical Web cluster is a pool of stand-alone server nodes, each of them being unaware of the others. The main performance limitation of this architecture is the shortage of memory resource on each server node, which forces it to retrieve most of the requested documents from its hard-disk: as this is often the bottleneck of a commodity-based server machine, the single node performance is bounded by the hard-disk performance. Yet, a Web cluster has a lot of aggregated RAM that can be used as a distributed cache to be accessed via a light-weight cooperation protocol, exploiting RAM-to-RAM transfers through the internal LAN, which can be two orders of magnitude faster than a local disk-to-RAM transfer: this is what we call a *global caching* architecture. To the best of our knowledge, commercial Web clusters do not adopt cache cooperation solutions, whereas some examples exist in the research community (see: [Li, 2001], [Liu, 2000], [Song, 2000]).

The logical topology of the cluster is considered to be a flat mesh. This choice enables us to design the cooperation protocol in a fully distributed and peer-to-peer fashion, as all the nodes are responsible of the same Web dominion. Distributed file systems are often used to share files within a Web cluster, because of their transparent interface toward the user-level applications. However, there are some performance trade-offs in this approach due to their architecture, designed to work in a read/write environment. Hence, unlike the distributed file systems, the caching cooperation protocol should run at the application level, because it needs to manage whole files, instead of disk blocks. Moreover, present disk storage dimensions allow us to replicate the whole Web content on the disk of each node, that simplifies file retrieval.

In the following sections, we discuss how the modules of CWebSim are instantiated to simulate some cooperation alternatives, and present some performance results.

3.1. The cooperation engine

We integrate a cooperation engine into the basic framework of CWebSim to simulate various GCP (Global Caching Protocol) schemes. This extension transforms the pool of stand-alone servers into a cluster of peer-to-peer cooperating servers. Only the Server module is directly affected by these modifications, while the remaining modules are instantiated over their original structure.

3.1.1. Server module extensions

Each server node is endowed with a cooperation frame consisting of a fully replicated Web content on its hard-disk, and of an engine, which has the following CSIM-based components (see figure 4: CPUs and NICs are not shown).

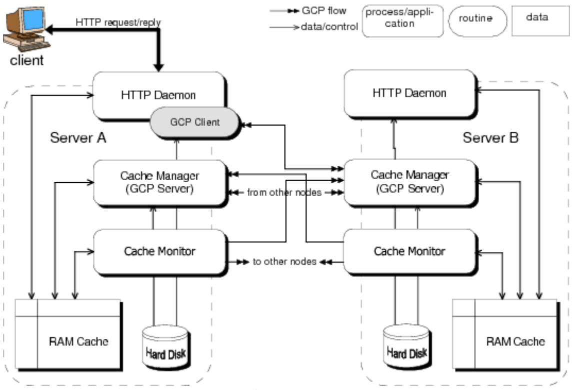


Figure 4: Model of the cooperation engines

RAM Cache: this is the local main memory storage of Web documents (i.e., a memory bank). The cache is managed with a LFU-Aging replacement policy (see: [Arlitt, 1997]) allowing a maximum age of 600 s, without any global coordination. Each HTTP daemon can access directly its local RAM Cache (through its CPU), whereas the other remote RAM Caches can be looked up via a “GCP Client Routine”. The RAM Cache is accessed for meta-information management (document lookup and location update) approximating simple hash-based operations, and for object insertion/replacement. The hashing CPU cost depends on the number of objects hosted in the cache, whereas the insertion/replacement CPU cost is proportional to the size of the target object.

GCP Client Routine: this procedure, embedded in the HTTP daemon’s code, plays the client role of the GCP, sending service requests to a remote “Cache Manager” and waiting for replies from it. The goal of these requests is to retrieve a Web document in a sibling cache, within a timeout sufficiently lower than the mean disk service time but not inferior to the round-trip time of the internal network.

Cache Manager: this CSIM process plays the server role in the GCP, accepting two kind of requests: service requests from remote GCP Clients (HTTP daemons), to which a reply must follow, and update issues from remote “Cache Monitors”, which trigger modifications on the local meta-information.

Cache Monitor: it is the CSIM process responsible of checking periodically the local meta-information in order to update the state of the global cache. Any change is broadcasted (update issue) to the other nodes; these messages do not require replies.

The HTTP request/reply stream is conveyed by the external network, whereas inter-node messages (GCP flow) are transmitted over the internal network. HTTP daemons and Cache Managers start refusing connections when their load conditions reach, respectively, 100 and 10 active connections.

The default computing flow of the simulated Web server (Section 2.1.3) is modified as follows: after a client request has been accepted and dispatched to a node, if the requested document is not found in the local cache (local cache miss), the GCP is activated before the hard-disk is accessed. This interaction scheme is based upon two sequential phases, *global lookup* and *document retrieval* .

In the **global lookup** phase, a cooperation policy aiming to provide the nodes with a global view of the system caches is applied. We focus on an adaptation of the *informed protocols* relying on periodic broadcast communication to update the global state of the system; other informed-based solutions for Web clusters are proposed in [Li, 2001] and [Ahn, 2000].

Document retrieval techniques consist of a set of operations that are carried out by the GCP to satisfy a Web request after a successful global lookup phase. These operations involve only two nodes: the first contacted node (A) and the node that has been selected by the global lookup step (B). We focus on two techniques:

Migration technique. The requested file is moved from the node B to the node A ([see: Ahn, 2000]). This mechanism balances the competition for global cache space, as the number of document replicas is unchanged, but it can consume lots of network bandwidth.

Handoff technique. Unlike the previous technique, the handoff mechanism requires kernel-level interventions to move the TCP connection to a different node, but it does not transfer any document (see: [Pai, 1998], [Song, 2000]). In this case, the node A "delegates" the node B to complete the Web transaction.

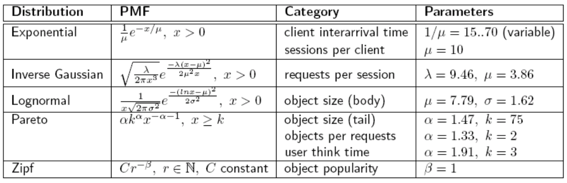
The hard-disk on the node A is accessed only when the global lookup fails (global cache miss).

3.1.2. Other simulation modules

Dispatching module. In the basic global caching cluster, the Web switch node is not involved in the cooperation protocol. In this case study, the Web switch is instantiated as a *one-way layer-4* device working with a *round-robin* selection algorithm: each incoming request is circularly assigned to a different server-node through manipulation of network-level packet headers. As this is one of the fastest dispatching mechanisms, it ensures that the Web switch is not a performance bottleneck: indeed, packets belonging to the same TCP connections are routed to the selected server by changing their MAC destination address (*packet forwarding*, see [Cardellini, 2002]); no further processing is done at the higher network layers.

Network module. For the interconnection of the cluster nodes, a 100 *Mb/s switched network* model is adopted, whereas the external network, which, simulating the Internet, connects the unique client node to the Web switch node and the server nodes to the client node is *ideal*. This configuration guarantees that all the cluster's ingress and egress points are not a performance bottleneck. The GCP is characterized by a minimum message size of 200 *B*; GCP requests timeout after 2 *ms* and metadata updates occur every 10 *s*. The simulated Web protocol is the HTTP/1.0.

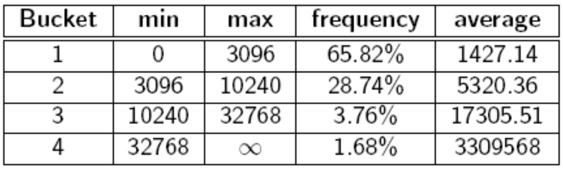
Input module. Table 3 summarizes the distributions adopted for the workload, together with their probability mass functions (PMFs) and parameters. The mean client inter-arrival time is the driver parameter of the entire test series. Note that the size distribution of Web objects (both HTML files and embedded objects) is hybrid, that is, the body and the tail are modeled by two different functions.



Distribution	PMF	Category	Parameters
Exponential	$\frac{1}{\mu}e^{-x/\mu},\ x>0$	client interarrival time sessions per client	$1/\mu = 15..70$ (variable) $\mu = 10$
Inverse Gaussian	$\sqrt{\frac{\lambda}{2\pi x^3}}e^{\frac{-\lambda(x-\mu)^2}{2\mu^2 x}},\ x>0$	requests per session	$\lambda = 9.46,\ \mu = 3.86$
Lognormal	$\frac{1}{x\sqrt{2\pi\sigma^2}}e^{\frac{-(lnx-\mu)^2}{2\sigma^2}},\ x>0$	object size (body)	$\mu = 7.79,\ \sigma = 1.62$
Pareto	$\alpha k^\alpha x^{-\alpha-1},\ x \geq k$	object size (tail) objects per requests user think time	$\alpha = 1.47,\ k = 75$ $\alpha = 1.33,\ k = 2$ $\alpha = 1.91,\ k = 3$
Zipf	$Cr^{-\beta},\ r \in \mathbb{N},\ C$ constant	object popularity	$\beta = 1$

Table 3: Workload distribution and parameters

The simulated Web content counts 5000 files and has an average file size of 57 *KB*, whereas the HTTP request stream exhibits a mean request size of 32 *KB* (different parameters are used for the request size distribution). Since the file size follows an heavy-tailed distribution, to give an idea of the working-set size, we may divide the distribution in 4 buckets as shown in table 4, where the *min*, *max* and *average* values are expressed in bytes. We refer to *working-set* as the total of all the files accessed during the observed period. From this file size discretization, by summing up the resulting amount in each bucket, we obtain a working-set size of about 280 *MB*.



Bucket	min	max	frequency	average
1	0	3096	65.82%	1427.14
2	3096	10240	28.74%	5320.36
3	10240	32768	3.76%	17305.51
4	32768	∞	1.68%	3309568

Table 4: A discretization of file size distribution

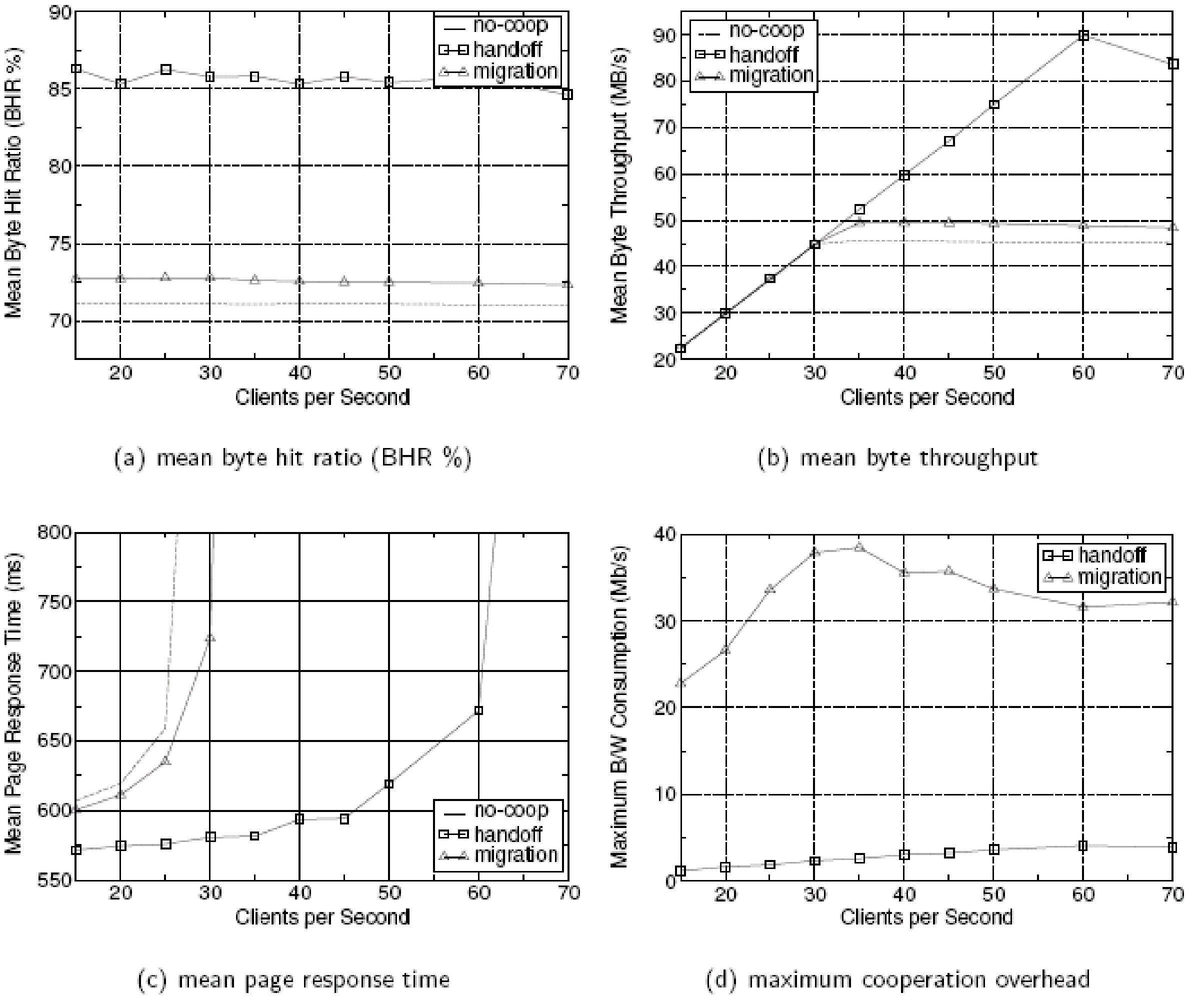
Sooner or later all files are requested at least once during the simulation, but the "core" of the most accessed ones in the working-set amounts to about 37 *MB*, because it contains many small objects. The composition of the core-set is determined by the Zipf distribution, that captures the locality of references in the whole Web content.

3.2. Performance study

This section presents a selection of the experimental work executed to evaluate the performance of the GCP in different Web scenarios. We focus on two *informed* cooperation schemes, adopting a *migration*-based and a *handoff*-based document retrieval mechanism, respectively; the reference system configuration is a non-cooperative cluster. Hereafter, we refer to the three cluster configurations as Migration, Handoff and No-coop.

The performance metrics we consider are the sample means of the *byte hit ratio* (*BHR*), of the *byte throughput* and of the *page response time*. Moreover, the *maximum B/W consumption* over the internal network is recorded to evaluate the peak cooperation overhead. All the reported values are the aggregate results of multiple experiments referring to the whole cluster.

The goal of this experiments is to evaluate the peak performance of the GCP schemes. The system under testing is a cluster with eight server-nodes, each of them endowed with 5 *MB* of cache: the global cache size is sufficient to hold the whole working-set's core. The mean number of incoming client process per second has been progressively increased from 15 to 70, until the saturation point of the best performing scheme has been reached. The system performance is evaluated under a Web scenario characterized by a static workload wherein all requested objects are cacheable. The results of this test series are reported in figure 5.



(a) mean byte hit ratio (BHR %)

(b) mean byte throughput

(c) mean page response time

(d) maximum cooperation overhead

Figure 5: Increasing load tests on an eight node cluster

As shown in figures 5(a), 5(b) and 5(c), to obtain a significant performance boost, it is necessary that a cooperative scheme yields a BHR improvement of at least 13-14%, with respect to the no-coop scheme. The poor performance of the Migration scheme is due to an excessive cooperation overhead, as shown in figure 5(d): the bandwidth consumption experiences a steep increase until it reaches a peak at 30-35 *clients/s*, that corresponds to the knee of the throughput and response time curves. On the other hand, the Handoff scheme transmits only control messages over the internal network, thus it can sustain higher loads.

4. SIMULATION OF WEB SWITCH DISPATCHING POLICIES

In cluster-based Web services the role of the Web switch node is fundamental to obtain adequate system performance. This section shows how CWebSim is instantiated to study the behavior of dispatching policies for dynamic Web-based services: we focus on the modules of the simulator that are affected by some modifications or by a different configuration with respect to the previous case study.

Dispatching module. The components of the Web switch node, shown in figure 6, are described below.

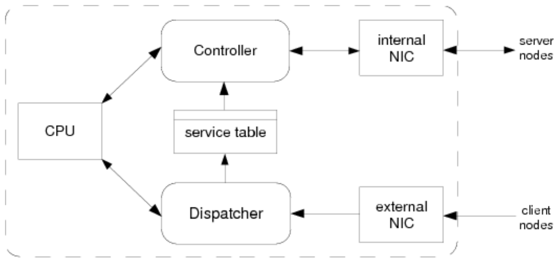


Figure 6: Web Switch node model

Dispatcher: this CSIM process simulates a kernel-level thread that receives the connection requests coming from the Web client nodes, through the external NIC (an ideal zero-delay queue). The first connection request of each session (see Section 2.1.1) is registered in the *Service table*, so that subsequent requests belonging to the same session are properly managed.

Service table: this data structure is used to relay session control information, needed by layer-7 dispatching algorithms, between the Dispatcher and the *Controller* processes. Conversely, layer-4 algorithms ignore that information.

Controller: this CSIM process simulates the kernel-level thread that executes the dispatching algorithm, basing its decision upon the data read from the Service table and upon some server-state information collected from the server nodes. Two content-aware selection criteria are considered: LARD (see: [Pai, 1998]) and CAP (see: [Casalicchio, 2001]); as a reference algorithm we consider the WRR (see: [Casalicchio, 2001]), which is a content-blind policy. The CAP policy, in its simplest form, does not require to set any parameter other than the choice of classes of services that can be identified from the URL. The LARD strategy requires to set only the parameters T_{low} and T_{high} indicating the minimum and maximum allowed server utilization. In our experiments we set T_{low}=0.3 and T_{high}=0.7, and we did not observed considerable changes for slightly different values, such as 0.2 and 0.75. According to the adopted dispatching algorithm, the CPU is used for a time ranging from some tenths to some hundreds of microseconds, every time a connection is accepted. Dispatched requests reach the selected server via the internal NIC.

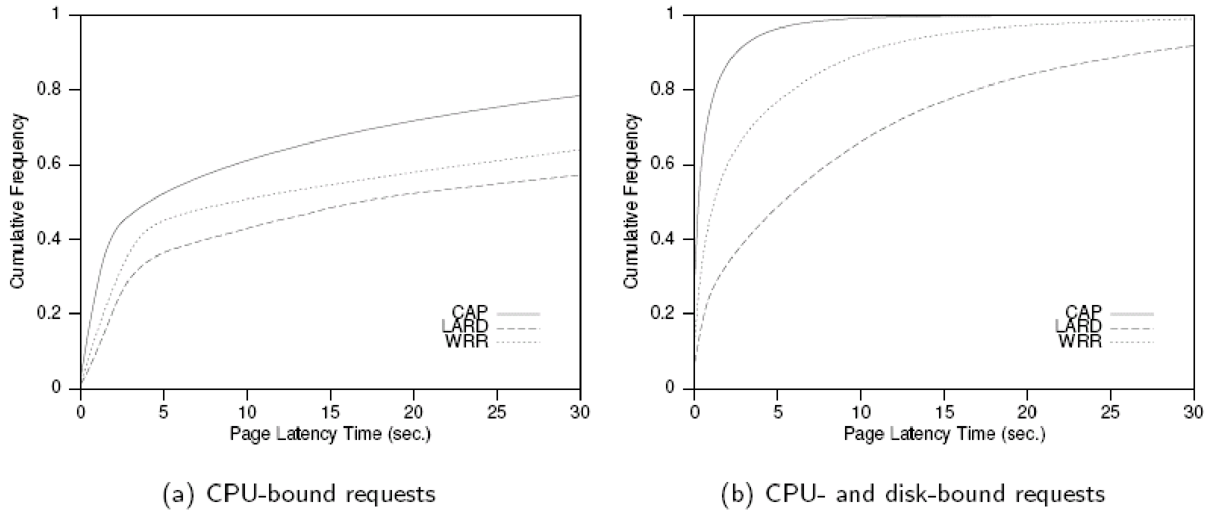
Server module. A basic server node model is adopted: the computing flow is the same described in Section 2.1.3 and no caching cooperation protocol is executed. Besides the HTTP Daemon, another CSIM thread simulates a periodical task that samples the node load state and transmits it to the Web switch over the internal network.

Input module. Because of the dynamic nature and security requirements of e-commerce transactions, the CWebSim's synthetic workload generator is configured to issue a certain share of dynamic and/or secure requests, according to various parameters. A ciphering algorithm is modeled via a single throughput parameter defining the CPU's processing speed. For example, we use 38.5 *Kb/s* for the 256-bit-key RSA, 46886 *Kb/s* for the Triple DES and 331034 *Kb/s* for the MD5. Secure requests are considered CPU-bound requests. The simulator allows us to define also dynamic requests that stress the server disk, namely, disk-bound requests.

4.1. Simulation results

The considered performance measure is the cumulative frequency of the *page latency time*.We consider three scenarios where light dynamic CPU- and disk-bound requests are mixed in different ways. In the first set of experiments, we assume that a 40% of the total requests need secure connections and cryptography. This is the most critical scenario because CPU-bound requests affect the performance of the other 60% of requests that have to use the CPU, even if for less intensive operations such as parsing a request and building an HTTP response. figure 7(a) shows how this very critical (and, we can say, unrealistic) scenario deteriorates the performance of all dispatching strategies. However, multiple service scheduling allows CAP to achieve better results than WRR and LARD.

In the second scenario, we introduce both CPU- and disk-bound requests, but CPU-bound requests (20%) differ from disk-bound requests (20%). In figure 7(b), we see that the CAP policy provides really good page latency times, while WRR reaches its limit and LARD does not guarantee a scalable Web cluster. The improvement of the CAP strategy is significant, especially if we consider that the WRR curve refers to a WRR policy with optimal parameter setting. For example, CAP achieves a page latency time of about 4 seconds with 0.95 probability. For WRR, a similar latency time is achieved with a probability of about 0.78, and for LARD with a probability of less than 0.5.



(a) CPU-bound requests

(b) CPU- and disk-bound requests

Figure 7: E-commerce scenario, cumulative frequency of page response time

5. RELATED WORK

Simulation frameworks oriented to the evaluation of network-based systems have a general scope and do not focus on Web cluster systems. Multiple general purpose simulators are available in the literature and in the industry. For example, the *Network Simulator ns-2* (see: [McCanne, 2007]) is an example of a discrete-event simulator strictly conceived for networking-related research, such as the testing of new communication protocols. Similar to the ns-2, are the *Georgia Tech Network Simulator GTNetS* (see: [Lee, 2005]) and *YANS* (see: [Lacage, 2006]). The Network Simulator is undergoing a new development cycle that will improve the general design, taking advantage from GTNetS and YANS: besides addressing scalability and code cleanup issues, the new *ns-3* (see: [Henderson, 2006]) aims at simulating low-level network details, without addressing explicitly Web clustering features. The *OPNET Modeler* (see: [OPNET, 2007]) is a comprehensive industry-standard simulation environment, that can be used to model communication networks and distributed systems: it incorporates tools for all the phases of a performance study, including graphical model design, simulation, data collection and analysis.

Another general purpose simulation and analysis tool is the *Tangram II* suite (see: [de Souza e Silva, 2006]), which allows the analysis of distributed systems using a combination of analytical and simulation models. However, like the other tools mentioned above, Tangram II is not specifically designed for the analysis of Web-based systems.

In the research area of Web systems, simulators are typically created for specific goals, that is, they cannot be instantiated over different target systems without significant modifications. For example, Cherkasova and Phaal model a single Web server as a service center with a finite listen queue, to investigate a particular admission control policy. Similarly, Cherkasova and Karlsson use a CSIM-based simulator that implements a model of a Web cluster: the goal of the authors is to evaluate the performance of a workload-aware request dispatching policy.

Pai et al. propose a Web cluster model wherein the Web switch is ideal, and each Web server is modeled through a waiting queue, simulating the NIC, and two service centers simulating the CPU and the hard-disk, respectively. Trace-driven simulations are executed to study the performance of a locality-aware dispatching policy.

A hybrid simulation tool is presented by Wang et al.: it integrates ns-2 with Logsim, a Web server simulator similar to that adopted in [Pai, 1998]. The goal is to explore the performance of request redirection mechanisms in Content Delivery Networks, that are geographically deployed systems. Similarly, the trace-driven discrete-event tool presented by Davison simulates geographic proxy-caching architectures, with efficient network-level models, though less detailed than those available in ns-2. Typically, the server-side of these simulators is not modeled with all the details required by the complexity of Web cluster systems.

6. CONCLUSIONS

This paper presents the design and implementation of a modular simulation framework (CWebSim) conceived for Web-based services deployed over clusters of HTTP servers. The basic structure of CWebSim can be configured to represent a broad variety of cluster-based Web architectures; also, it is possible to build custom system models through extensions to its functional modules. Particular attention has been dedicated to the network and workload models, which are customizable with different degrees of realism to allow simulations of the most typical Web scenarios. The workload module is capable of both distribution- and trace-driven generation modes.

CWebSim is written in C and it is based on the CSIM library, which is available for the most popular platforms, including Linux and Windows.

We have presented a first case study where CWebSim is used to simulate global caching systems: the simulation of these architectures required a careful modeling of the server-to-server interaction occurring through the internal network. In a second case study, we focus on the simulation of diverse Web switch dispatching algorithms, that required a detailed modeling of the dispatcher components.

CWebSim is being extended to support the modeling of low-level network-related issues, so as to allow the simulation of geographically distributed Web systems.

REFERENCES

W. H. Ahn, S. H. Park, and D. Park (2000). “Efficient Cooperative Caching for File Systems in Cluster-based Web Servers”. In Proc. of the 2nd IEEE Int’l Conf. on Cluster Computing (CLUSTER2000)

M. F. Arlitt and T. Jin (2000). “A Workload Characterization Study of the 1998 World Cup Web Site”. IEEE Network, 14(3):30–37.

M. F. Arlitt and C. L.Williamson (1997). “Internet Web servers: Workload Characterization and Performance Implications”. IEEE/ACM Trans. on Networking , 5(5):631–645.

M. F. Arlitt and C. L. Williamson (1997). “Trace-driven Simulation of Document Caching Strategies for Internet Web Servers”. SIMULATION Journal , 68(1):23–33.

P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella (1999). “Changes in Web Client Access Patterns: Characteristics and Caching Implications”. World Wide Web , 2(1-2):15–28.

P. Barford and M. E. Crovella (1998). “Generating Representative Web Workloads for Network and Server Performance Evaluation”. In Proc. of the ACM Performance 1998/SIGMETRICS 1998 , pp. 151–160, Madison, WI, USA.

P. Barford and M. E. Crovella (1999). “A Performance Evaluation of Hyper Text Transfer Protocols”. In Proc. of the ACM SIGMETRICS 1999 , pp. 188–197, Atlanta, GA, USA.

V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu (2002). “The State of the Art in Locally Distributed Web-server Systems”. ACM Computing Surveys , 34(2):1–49.

E. V. Carrera and R. Bianchini . “Efficiency vs. Portability in Cluster-based Network Servers”. In Proc. of the 8th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP’01) , pp. 113–122, Snowbird, UT, USA, June 2001.

E. Casalicchio and M. Colajanni (2001). “A Client-aware Dispatching Algorithm for Web Clusters Providing Multiple Services”. In Proc. of the 10th Int’l World Wide Web Conf. (WWW’01) , Hong Kong, HKG.

L. Cherkasova and M. Karlsson (2001). “ScalableWeb Server Cluster Design withWorkload-aware Request Distribution Strategy WARD”. In Proc. of the 3rd Int’l Work. on Advanced Issues of E-commerce and Web-based Information Systems , San Jose, CA, USA, June 2001.

L. Cherkasova and P. Phaal (1999). “Session Based Admission Control: a Mechanism for Improving Performance of Commercial Web Sites”. In Proc. Int’l Work. on Quality of Service , London.

M. E. Crovella and A. Bestavros (1997). “Self-similarity in World Wide Web Traffic: Evidence and Possible Causes”. IEEE/ACM Trans. on Networking , 5(6):835–846.

B. D. Davison (2001). “NCS: Network and Cache Simulator – An Introduction”. Technical Report DCS-TR-444, Rutgers University, Department of Computer Science.

T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley (2006). “ns-3 Project Goals”. In Proc. of the Workshop on NS-2 2006 , Pisa, IT.

M. Lacage, and T. R. Henderson (2006). “Yet Another Network Simulator”. In Proc. of the Workshop on NS-2 2006 , Pisa, IT, Oct. 2006.

Y. J. Lee, and G. F. Riley (2005). “Efficient Simulation of Wireless Networks using Lazy MAC State Update”. In ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS).

Q. Li and B. Moon (2001). “Distributed Cooperative Apache Web Server”. In Proc. of the 10th Int’l World Wide Web Conf. (WWW’01) , Hong Kong, HKG.

W. Liu, M. Wu, W. Zheng, and M. Sheng (2000). “Design of an I/O Balancing File System on Web Server Clusters”. In Proc. of the 2000 Int’l Work. on Scalable Web Services (in conjunction with ICPP’00) , Toronto, ON, CAN.

S .McCanne, and S. Floyd (2007). “ns-2 Network Simulator”. http://www.isi.edu/nsnam/ns/.

Mesquite Software, Inc. (2007) “CSIM”. http://www.mesquite.com/.

OPNET Technologies, Inc. (207) “OPNET Modeler”. http://www.opnet.com/.

V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. M. Nahum (1998). “Locality-aware Request Distribution in Cluster-based Network Servers”. In Proc. of the ASPLOS-VIII, San Jose, CA, USA

J. E. Pitkow (1999). “Summary of WWW Characterizations”. World Wide Web, 2(1-2):3–13.

J. Song, E. Levy-Abegnoli, A. Iyengar, and D. Dias (2000). “Design Alternatives for Scalable Web Server Accelerators”. In Proc. of the 2000 IEEE Int’l Symp. on Performance Analysis of Systems and Software, pp. 184–192, Austin, TX, USA.

E. de Souza e Silva, R. M. M. Lenao, A. P. Couto da Silva, A. A. A. Rocha, F. P. Duarte, F J. Silveira Filho, G. D. G. Jaime. and R. R. Muntz (2006). “Modeling, Analysis, Measurement and Experimentation with the Tangram-II Integrated Environment”. In Proc. of ValueTools 2006, Pisa, IT.

L. Wang, V. Pai, and L. Peterson (2002). “The Effectiveness of Request Redirection on CDNRobustness”. Technical Report TR–654–02, Princeton University.

AUTHORS BIOGRAPHIES:

Dr. Emiliano Casalicchio is with the Computer Science Systems and Production Department of the University of Roma "Tor Vergata". His main research interests are on distributed systems design modeling and performance evaluation, with a particular interest on Grid Computing, Web Systems and Technologies for wired and mobile environments. Since 2004 he has been also working on Interdependency Analysis in Critical Information Infrastructures, working on Agent-based modeling and simulation of complex systems.

He acts as reviewer for international conferences, and journals: IEEE Transaction on Software Engineering (2007), Computer Communication Journal, Elsevier (2007); IEEE Internet Computer(2005); Journal of Systems and Software Elsevier (2004); Performance Evaluation Review, NorthHolland 2003; IEEE International Conference on Communication 2003; IEEE Vehicular Technology Conference 2004 - Fall; IEEE Vehicular Technology Conference 2003; ACM-WIAPP2003; ICC2003. He is an IT consultant for many Italian companies and for the Italian Custom Agency and he is actually a member of the editorial committee of the Italian Society of Critical Infrastructure Experts (AIIC).

Dr. Riccardo Lancellotti received the Laurea and the Ph.D. degrees in computer engineering from the University of Modena and from the University of Rome “Tor Vergata”, respectively. He is currently an Assitant professor in the department of Information Engineering at the University of Modena, Italy. In 2003 he spent eight months at the IBM T.J. Watson Research Center as a visiting researcher. His research interests include scalable architectures for Web content delivery and adaptation, peer-to-peer systems, distributed systems performance evaluation and benchmarking. He is a member of the IEEE Computer Society. For additional details see
http://weblab.ing.unimo.it/people/riccardo.

Dr. Marco Emilio Poleggi received his MSc in Electronic Engineering from University of Rome "La Sapienza" and his PhD in Computer Engineering from the University of Rome "Tor Vergata". During his studies, he focused on cache-cooperation mechanisms for cluster-based Web servers. He currently works at CERN – the European Organization for Nuclear Research - as a research fellow: he is a developer and the release manager of Quattor, an open source tool suite for the administration of large computer clusters. Marco Emilio is a member of the IEEE Computer Society. Contact him at <Marco.Poleggi@cern.ch>