Impact of technology trends on the performance of current and future Web-based systems

Mauro Andreolini[1], Michele Colajanni[1] and Riccardo Lancellotti[1]

[1]University of Modena and Reggio Emilia,
Department of Information Engineering
Via Vignolese 905, 41100 Modena, Italy
{andreolini.mauro,colajanni,lancellotti.riccardo}@unimo.it

Abstract: **The hardware technology continues to improve at a considerable rate. Besides the Moore law increments of the CPU speed, in the last years the capacity of the main memory is increasing at an even more impressive rate. One of the consequences of a continuous increment of the main memory resources is the possibility of designing and implementing memory-embedded Web sites in the near future, where both the static resources and the database information is kept in the main memory of the server machines. In this paper, we evaluate the impact of memory and network technology trends on the performance of e-commerce sites that continue to be an important reference for Web-based services in terms of complexity of the hardware/software technology and in terms of performance, availability and scalability requirements. However, most of the achieved considerations can be easily extended to other Web-based services. We demonstrate through experiments on a real system how the system bottlenecks change depending on the amount of memory that is (or will be) available for storing the information of a Web site, taking or not into account the effects of a WAN. This analysis allows us to anticipate some indications about the interventions on the hardware/software components that could improve the capacity of present and future Web-based services.**

Keywords: Web-based services, system architecture, performance evaluation, bottleneck analysis, client/server**.**

I. Introduction

Web technology is the standard interface towards many services that are exploited through the Internet. The demand for better service performance has driven a technological trend towards more powerful, integrated, scalable system components. The capacity of current hardware components such as processor, main memory, and network interfaces is continuously growing. The main interest of the scientific community has always been focused on the impressive growth of the CPU power that continues to follow the Moore law [18], although the asymptotic bound seems closer. This interest has, in a certain measure, masked the fact that, in the last years, the capacity improvements of the main memory and the network capacity are even more impressive than the CPU increases. The combination of these multiple hardware resource improvements is having or will have important implications on the performance of the Internet-based services. In the traditional client-network-CPU-disk path, the adoption of large main memories, together with CPU speed improvements, seems a valuable answer to the continuous research of bottleneck removals that often occur at the disk side. The trend of adopting large central memories will help the user to experiment lower response times, but it has the potential to create novel difficulties to the server component of an interactive Web-based service [21]. If we limit our considerations to Web-based services that are of interest for this paper, we can observe that the size of a typical Web site in terms of stored information tends to grow slower than the typical size of the main memory. The reader should consider that nowadays some Gbytes of RAM are the entry level of any server machine that supports interactive services with some performance requirements [8]. And, in the large majority of cases, the most requested information of typical Web sites that are implemented through a multi-tier architecture does not span over some Gbytes of data stored in file systems and databases. Hence, it seems practicable to foresee the design and implementation of so called *memory-embedded Web sites*, where both the static resources and the database information is kept entirely in the main memory of the server machines.

If we pass to consider the network capacity, we can observe bandwidth improvements at many levels, from the *first mile* (that is, the Internet connection of the Web site servers), to the *backbones* of the Internet core (excluding peering points among Autonomous Systems that remain an open issue), to the *last mile* where a larger percentage of final users are adopting fast connection technologies (especially, ADSL and cable connections) [23]. These network improvements will certainly have further impacts on the architectures and performance of the future Web-based services.

Some consequences of these technological trends are intuitive. For example, we can easily expect that the system capacity will tend to increase and any Web-based services will be able to guarantee higher throughputs. Other consequences are not fully exploited. For example, it is unclear which will be the new bottlenecks that will bound the maximum capacity of a Web-based system. In this paper, we aim to answer to

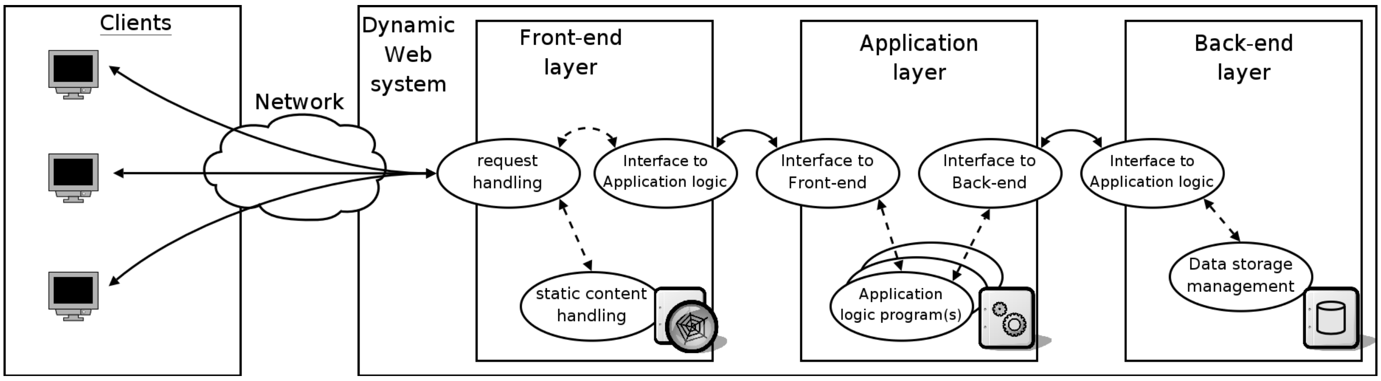


Figure 1. Architecture of a dynamic Web site

some of these questions by evaluating the impact of hardware improvement trends, especially main memory and network capacity, on the performance of e-commerce sites. These types of Web-based services are important because they will continue to represent examples of mission critical Internet-based services, in terms of complexity of the hardware and software technology and in terms of performance, availability and scalability requirements. Through endurance and stress tests, and accurate measurement of coarse grain and fine grain performance parameters, we show how the system bottlenecks of a prototype e-commerce site will change as a function of the amount of available main memory and the quality of the interconnection between the clients and the server. Our trend analysis has important consequences because it allows to anticipate some interventions on the software components to further improve the capacity of present and future Web-based services. The three main contributions of this paper are summarized below.

First, we confirm the intuition that memory-embedded Web sites have lower response times and better throughput than sites that keep a significant portion of static and dynamic information on disk. The obvious reason behind the different performance is the speed of current storage areas, which is really slow when compared to the main memory accesses. The performance gap between disks and main memory becomes much more evident if we think that every source of static content has to be fetched from a file system and almost every source for dynamic content requires information that is usually fetched from a database server.

Second, for all the considered workload models, we verify that the back-end node hosting the database server remains the system bottleneck. However, a fine grain analysis of the system resources of this back-end node allows us to evaluate how the bottlenecks change as a function of available amount of main memory. We distinguish three results. For an almost memory-embedded database, disk accesses are rare, and the bottleneck is represented by the CPU of the database server because of synchronization operations with in-memory data buffers. When a significant portion of the database (about 50%) is kept in main memory, the system capacity is typically limited by the number of available socket descriptors, that are a pool of limited resources. When a small part of the database is kept in RAM, the disk operations represent the bottleneck reducing the system capacity.

Third, we evaluate the impact of wide area network effects on the performance of the considered e-commerce site. We find out that, when the client-system connections are slow, the duration of the user session augments, and long lasting connections tend to exhaust the pool of available socket descriptors. This may have tragic consequences on the system performance because descriptors are token-based limited resources that do not degrade gracefully such as other system resources (e.g., CPU). On the other hand, when the client-system connections improve, the system bottleneck tends to move towards the CPU.

The rest of this paper is organized as following. Section 2 outlines the main functions of the components of a representative Web-based dynamic site of medium size and average popularity. Section 3 illustrates the dynamics of a request service and points out possible performance problems. Section 4 describes the testing plan that is pursued during the experiments. Section 5 describes the experimental testbed and the workload models that are used for the experiments. Section 6 analyzes the experimental results about the performance impact of memory technological trends, while Section 7 focuses on the impact of network bandwidth features. Section 8 discusses some related work. Finally, Section 9 concludes the paper with some final remarks.

II. System architectures for Web-based services

After some years of initial innovation and confusion, today's basic architectures for building Web systems share a core set of basic design choices. These choices can be applied to Web sites providing a mix of static and dynamic content, as well as Web services that provide interaction between information systems through a Web interface. Throughout the rest of the paper, we will use the term *Web-based service* to address both Web services and static/dynamic Web sites. In other words, a Web-based service is considered any Internet service that uses an HTTP-based interface.

Independently of the large variety of available software solutions, the typical design of a Web-based service is nowadays based on a multi-tier logical architecture that tends to separate the three main functions of service delivery: the HTTP interface, the application (or business) logic and the information repository. These logical architecture layers are often referred to as *front-end*, *application*, and *back-end* layers [2]. Figure 1 shows the main structure of a typical system providing Web-based services. Beside the clients, we recognize the three logical components of the system (shown as boxes), each detailed with its main functions (the ovals within the boxes). This figure also sketches the fundamental interactions between the three layers (with solid lines) and between the main functions provided by each layer (with dashed lines).

The *front-end* layer is the interface of the Web-based service. It accepts HTTP connection requests from the clients, serves static content from the file system, and represents an interface towards the application logic of the middle layer (as shown in Figure 1). The most popular software for implementing the front-end layer is the Apache Web server, although others exists, e.g., MS Internet Information Services, Sun Java System Web Server, Zeus.

The *application layer* is at the heart of a Web-based service: it handles all the business logic and computes the information which is used to build responses with dynamically generated content. Content generation often requires interactions with the back-end layer, hence the application layer must be capable of interfacing the application logic with the data storage of the back-end. There is a huge number of technologies for deploying a business logic on the middle tier. CGI has been the first software technology for the deployment of dynamic Web sites, but the performance penalty related to the creation of a CGI process for every client request has led to the decline of the CGI popularity. Scripting languages, such as PHP and ASP, are quite popular for medium size dynamic Web sites. Component-based technologies belonging to the J2EE family (such as Java Servlets, JSP and EJB) are considered more scalable [7] and are often preferred for building medium-to-large Web sites. The modularity of component-based technologies also makes the deployment of Web services easier. Indeed, most Web services are designed with reusability and composition of basic service facilities in mind: these concepts are best put into practice through the adoption of modular (object oriented) design techniques.

The *back-end layer* manages the main information repository of a Web-based service. It typically consists of a database server and storage of critical information that is the main source for generating dynamic content. Database servers have a long history and there are many good alternatives. Nowadays, popular frameworks are provided by Oracle, IBM DB2, MS SQL Server on the proprietary side, and by MySQL and PostgreSQL on the open source side.

A complete implementation of a Web-based service requires the logical design to be followed by an architectural design. In this latter phase, the multi-tier logical layers must be mapped on a physical architecture. Even in this phase, there are many alternative solutions, but the *cluster system* is becoming the most common choice for the supporting platform. A cluster consists of multiple nodes that are located in the same area network, and a single component that provides the interface with the external world. For the most popular Web-based services, the nodes may be distributed over a geographical area, but in this paper we do not consider similar architectures.

It is important to observe that there is not a one-to-one correspondence between the logical layers and the physical architecture: multiple layers may be mapped on the same node; alternatively, the same logical layer may be implemented on multiple nodes. For example, the software components (front-end, application and back-end servers) may run on one physical node or on a cluster of nodes [6]. The best choice depends on many factors, such as the adopted technology, the size and the popularity of the Web-based service, other security goals and economic constraints. If we refer to medium size Web sites, the real choice is between two or three physical nodes, because the common tendency is to map always the information repository on a separate node. With present technologies, a software such as J2EE [16] would lead to a physical separation of the three logical layers on at least three nodes. On the other hand, scripting technologies, such as ASP, JSP and PHP, tend to concentrate the front-end server and the application server on the same node.

Independently of the mapping choice, the software components of the Web system are strictly correlated. For example, the application server relies on the back-end layer to provide the necessary information for building the application logic data. If the back-end layer fails or results too slow, the performance of the application server may be severely degraded and, as a domino effect, the overall performance of serving dynamic requests drops. Each component consumes system resources, such as CPU power, main memory, file and socket descriptors. These resources are not unlimited, and usually they get exhausted when the system is subject to high load. The amount of available resources is a key factor in the performance of a dynamic Web-based service. When the amount of available resources varies (for example, a consistent amount of main memory is added to the system), the performance of the overall Web system changes. More importantly, the system capacity may be determined by a resource bottleneck that is a function of the current software and hardware layout of the system. Since hardware upgrades are becoming much more frequent (mainly due to the low costs of today’s off-the-shelf components), the service infrastructure is subject to a continuous evolution, which typically follows the current technology trends. We will show in this paper that it is not obvious to anticipate the new bottlenecks introduced by the adoption of new (or even future) technologies.

In particular, we focus our attention on the technology trends that characterize the availability of main memory at the server machines, and the continuous improvements occurring at the network level.

The cost of the main memory has been constantly decreasing over years and even entry level computers are now equipped with amounts of RAM that were found just in top level workstations just a few years ago. If we consider the impact that this trend is having and will have on the information repository for many Web-based services, we can foresee a radical shift. Most Web-based services are characterized by an amount of information seldom exceeding some Gbytes and the growth of data does not tend to follow the exponential growth of main memory availability. Indeed, over the years, the amount of main memory installed on servers hosting the information repository has been steadily increasing. As a consequence many Web sites are characterized by a DBMS that can store the whole database into RAM [19] and this fact is destined to become more common in the near future.

Besides the obvious performance gain due to the low access time of RAM if compared to disk, a RAM-based database can dramatically alter the performance of the Web system by shifting potential bottlenecks on multiple parts of the Web system.

In a similar way, wide area network technologies have dramatically increased the available bandwidth over geographic links, thus reducing the download time also for rich media content. Besides some performance gains that may be easily expected, in this paper we are mainly interested to evaluate the impact of the technology evolutions on the internal components of a system supporting a Web-based service.

III. Request service and potential bottlenecks

Answering to a request reaching a multi-tier Web-based system is a complex task that triggers the creation and interaction of several processes. In this section we will refer to Figure 1 for outlining the main dynamics behind the service of a client request.

After receiving a request, the request handling module at the front-end layer invokes the most suitable function for its service. The interactions are represented in Figure 1 as dashed lines connecting the request handling module with the other modules of the front-end layer. Requests for static Web objects can be satisfied by the static content handling subsystem at the front-end and do not usually put any significant load on the system. Indeed, serving static content typically requires the retrieval of one or multiple objects from the file system, which is a low overhead operation especially when the requested file hits the disk cache. The network adapter connecting the Web infrastructure to the outside world is the only system component that could be affected by the service of heavy static contents.

On the other hand, dynamic requests are passed by the front-end to the application layer through the proper interface module. The interaction between the two layers is shown as a solid line connecting the interface modules in Figure 1. The application layer generates the content through the cooperation with the back-end tier. The interface with the front-end activates a set of modules implementing the actual application logic. The process of activating the modules is denoted by the dashed line between the interface and the application logic subsystem. These modules are, typically, CPU-bound. To fulfil the client request, they can also request additional information to the back-end layer through the proper interface. The interaction between the layers is represented by a solid line in Figure 1.

When the DBMS is placed on a different machine, a communication between the application server and the DBMS requires the use of connection descriptors (that is, *sockets*). These are critical resources of the operating system, because they are a limited set.

The DBMS can place a significant amount of stress on different hardware resources of the server machine depending on the type of required operations. For example, the CPU may be loaded by operations related to a complex query, the disk may be loaded by operations that require many accesses to the mass storage. It is worth noting the different behavior of the system resources.

Sockets are *token-based* resources, which means that only a finite number of sockets is available and can be assigned to processes. When the available tokens are exhausted, additional requests are queued for an unpredictable time (i.e., until a token becomes free). A connection request may fail because the time-out deadline is passed or because it cannot be stored in the finite-length waiting queue of the service node.

On the other hand, the server CPU and disk are *gracefully degradable resources*, that may be shared among every process requesting them. Once the resource is fully utilized, additional requests lead to progressive performance degradation, but typically no waiting request is refused but in really critical situations.

IV. Performance testing plan

In this paper we aim to evaluate the effects of technological trends on the overall performance of Web-based services, and also to verify whether these trends may modify the bottlenecks that determine the maximum capacity of the system that supports Web-based services. The goal of the performance study determines the approach and the testing plan used for the experimental analysis.

We use a so called *black-box* testing to evaluate the performance of the system and to determine under which workload conditions the system reaches its maximum capacity. We also use a so called *white-box* testing when we aim to identify the hardware or software resource that represents the system bottleneck.

In a black-box testing, the system is viewed as an atomic entity, hence we consider its behavior as it is seen from the outside. The considered performance indexes are at the *system* level. Popular indexes are the response time, and the throughput that can be measured in terms of served requests (e.g., pages, hits) per second or Mbytes per second. The main goal of these performance indexes is to verify whether the system is providing or not services at an acceptable level of performance. Black-box testing is also useful for detecting the trends of system performance as a function of the offered load. It is a common practice to build a load curve for the Web system by considering, for example, the average (or, better, the 90-percentile) of the response time as a function of increasing request loads. Such a curve allows us to identify a knee region in which a sudden growth of the response time occurs. This region denotes that the system is working at its maximum capacity and at least one of its resources is critically loaded. Black-box testing does not consider the internal components of the system, hence it is impossible to identify the bottleneck that limits the system capacity.

When we have the necessity of a more accurate analysis, we carry out a white-box testing. In this case, we exercise the Web system with a client request load around the knee region that has been identified through the previous black-box testing, and monitor continuously the status of the internal resources of the Web system. This type of analysis requires the evaluation of finer grain performance indexes that are related to the internal resources of the Web system. We typically consider the performance at least at the *resource level* that are associated to the most important hardware and software (mostly, operating system) components. Some examples include: the utilization of the CPU, disk and network interface, or the amount of free memory at the hardware level; the number of available file and socket descriptors at the level of the operating system.

In particular, in this study we take into account cumulative distributions or percentiles instead of average values related to the previous performance indexes. The motivation comes from the observation that higher moments are necessary when we have to evaluate the performance of systems, such as e-commerce sites, subject to load arrivals that are characterized by heavy-tailed distributions [2].

V. Setup of the experiments

A. *Experimental testbed*

We carried out the entire set of experiments by considering an e-commerce site as representative Web-based service. The e-commerce site is implemented by a three-tier logical architecture that is mapped on a physical architecture consisting of two server machines. Figure 2 illustrates the architecture of the prototype system that has been used for the experiments.

The first node of the Web system hosts the software related to the front-end and the middle tiers. In particular we use the Apache Web server [4] for the front-end, and the PHP4 [22] engine to implement the application logic at the middle tier. The second node hosts the software for the back-end tier. We choose MySQL [20] as the database server. To reflect a realistic workload scenario, we enabled the support for table locking and two phase commits.

Other machines host the software running the client emulator, which is used to generate the appropriate volume of user requests to the Web system.

Each machine of the testbed platform has a 2.4 GHz hyperthreaded Xeon CPU and is part of a cluster of nodes that are connected through a Fast Ethernet LAN. Each node is equipped with the standard Linux operating system (kernel version 2.6.8), and monitoring tools to collect samples of performance measures that are necessary for the white-box testing. In particular, we use the *system activity report* [11] tool to collect resource utilization statistics. This tool samples at regular intervals the utilization of both physical resources such as CPU, memory and disk, and operating system resources, such as the number of open sockets and the number of processes. The output of the system monitor is logged for off-line analyses.

Since one of the main interests of this study is to evaluate the impact that the availability growth of the main memory may have on performance and system bottlenecks, we emulate three main *memory scenarios*:

- **All in-memory.** This scenario represents what we

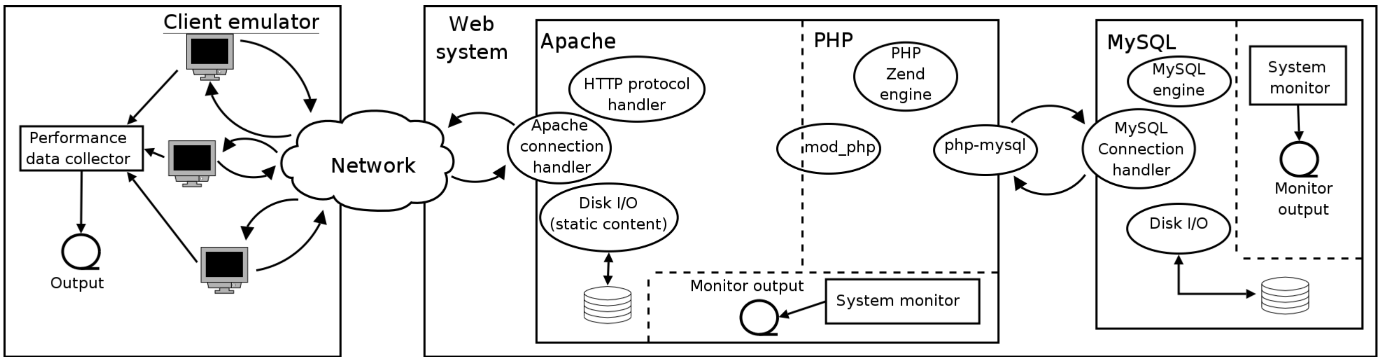


Figure 2. Experimental testbed

consider the next future, when the whole database information is likely to fit into main memory.

- **Partially in-memory.** This scenario represents a common present case, where just half of the database information can fit into main memory.
- **Mostly on-disk.** This scenario represents a past situation, where the main memory can host only a small portion of the database information.

For each scenario, Table I reports the percentage details of database information that can be stored into the main memory.

We also take into account the evolution of networks where the current trend corresponds to a general increment of the available bandwidth together with a reduction of the latency. To evaluate the impact of this trend on Web system performance and possible bottlenecks, we emulate different network scenarios that present a better or worse quality of the connection between the client emulator and the Web system.

Scenario	Available RAM [% of DB size]
All in-memory	100 %
Partially in-memory	60 %
Mostly on-disk	30 %

Table 1. Memory scenarios

Without the intention and the possibility of recreating the entire spectrum of Internet effects, we introduce some typical wide area network effects in the links connecting the client emulators to the node hosting the front-end layer: packet delay, loss and bandwidth limitation. To this purpose, we utilize a WAN emulator that is based on the *netem* packet scheduler of the Linux kernel [13]. In this paper, we consider two network scenarios:

- *Well connected clients.* We configure the WAN emulator by using a maximum link bandwidth of 64 Mbit/s and by introducing a packet delay with a normal distribution with μ = 5ms and σ = 1ms, with no packet loss.
- *Badly connected clients.* We configure the WAN emulator by creating a virtual link between the clients and Web system with the following characteristics: 8 Mbit/sec as the maximum link bandwidth; packet delay normally distributed with μ=200ms and σ=10ms; packet drop probability set to 1%.

B. *Workload model*

The choice of a realistic or at least adequate workload model to test an e-commerce system is an open problem by itself. In workload models oriented to browsing, the interaction is basically with the HTTP server and the mix is mainly oriented to define the number and size of embedded objects together with the user think time. However, in an e-commerce service we have dozen of possible alternatives at any level of the multi-tier architecture, often also dependent on the adopted software technology. As a consequence, it is impossible to define THE model for e-commerce services. A popular choice for the research community that has no access to actual logs of highly popular e-commerce sites is the use of the TPC-W benchmarking model. Together with the more recent SPEC-WEB 2005 [25], TPC-W represents the only complete specification (workload side and system side) of an e-commerce site that has been proposed and shared by a large scientific community [1, 10]. For this reason, a TPC-W like workload model is the choice of this paper.

In our experiments, the workload mix is composed by 5% of requests for static resources and 95% of requests for dynamic resources.

Web traffic is generated by means of a TPC-W like *client emulator*, which creates a fixed number of client processes. Each process instantiates sessions consisting of multiple requests to the e-commerce system. For each customer session, the client emulator opens a persistent HTTP connection to the Web server which lasts until the end of the session. Session length has a mean value of 15 minutes. Before initiating the next request, each emulated client waits for a specified think time, which is set to 7 seconds on average. The sequence of requests is determined through a state transition diagram that specifies the probability to pass from one Web request to another one.

The standard user behavior is to start its interaction with the Web site from the home page. The user may use a search function to select an item to purchase or browse products by category. Once an item has been selected, the user may browse a product description and place the related item in the shopping cart. Finally, the user may checkout the items in the shopping cart through a purchase transaction. This description represents a typical user interaction, but more complex user behaviors (such as aborting a transaction or removing items from the shopping cart) may occur, even if with a lower probability. A complete description of the state transition diagram of the user behavior is reported in [1].

VI. Performance impact of memory capacity

A. *Bottleneck identification*

In this section we analyze to what extent the technological trends concerning main memory may influence the performance of a Web-based service and may shift the bottlenecks across different components and resources of the Web architecture.

We first evaluate the response time for the three memory scenarios when the Web system is subject to increasing loads. The goal is to understand, for each of the proposed scenarios, the request arrival rates at which the Web architecture starts to evidence a clear bottleneck. Figure 3 shows the 90-

percentile of the response time as a function of the client population for the three memory scenarios. A comparison among the three curves shows that the impact of available main memory on the response times is much less significant than the impact on system capacity. The scarce influence of the main memory on response times is visible in Figure 3, that reports the 90 percentile of Web object response times as a function of user population. The main motivation is that, when the system is not overloaded, the amount of necessary main memory does not exceed the physical limits of the corresponding node, and no (slow) virtual memory management takes place. As a consequence, the response time is bound by the following operations: Web page construction (mainly CPU-bound), access to storage facilities (DISK-bound), and use of network resources for communication. This confirms the intuition that additional main memory on the back-end node increments the system capacity. We now want to investigate the reasons behind the impact of the main memory size on the maximum capacity of the Web system. To this purpose, for each memory scenario, we first identify a critical value of the client population that leads to a sudden degradation of the system performance: this corresponds to the so called *knee* of the response time curve. As we proceed from the mostly on-disk scenario (called simply *on-disk* in the following of this section) to the partially in-memory to the all in-memory scenario (*in-memory*), from Figure 3 we can see that the maximum number of clients that can be served without significant performance degradation increases from 30 to 270 to 330, respectively.

Once found the critical load that determines the maximum system capacity for each memory scenario, it is interesting to investigate the nature of the bottlenecks that limit the system capacity. This analysis requires a deeper comprehension of the internal behavior of the system components, that we have called white-box testing.

In a Web architecture consisting of multiple layers, the first step requires the identification of the layer causing the system bottleneck. The standard procedure is to split the contribution to the response time among its components that are related to the front-end, the application and the back-end layers. As example, Figure 4 shows the breakdown of the response time for the partially in-memory scenario. The histograms represent the 90-percentile of the three contributions for two population values. The bar on the left is related to a number of clients lower than the system capacity (equal to 270 users), while the bar on the right reports the breakdown for a client population higher than the system capacity.

A comparison between the two histograms shows that when the critical number of requests is reached, there is a sudden increase of the response time contribution referring to the back-end layer. Similar studies with the other memory scenarios confirm these conclusions. In all experiments, the increase in the back-end time drives the performance degradation of the system shown in Figure 3 for the considered TPC-W workload model. Hence, we can easily conclude that the bottleneck is always related to the operations at the database server node.

B. *Resource-level analysis*

After the identification of the maximum capacity of the Web architecture in terms of number of concurrent users (in correspondence of the knee of the curves), we pass to analyze the causes behind the bottlenecks that limit the capacity of the system. To this purpose, we carry out a *white-box* performance testing for the three memory scenarios and a population of clients in proximity of the identified knees. For the on-disk scenario we consider a population of 60 clients; for the partially in-memory scenario we consider a population of 300 clients, and for the in-memory scenario we consider a population of 360 clients. For each experiment, we monitor accurately the utilization of the main system resources. Recalling the dynamics behind a client request detailed in Section 3, we conclude that the main performance indexes that may evidence a system bottleneck are: CPU utilization, disk I/O activity (in terms of disk transactions per second)

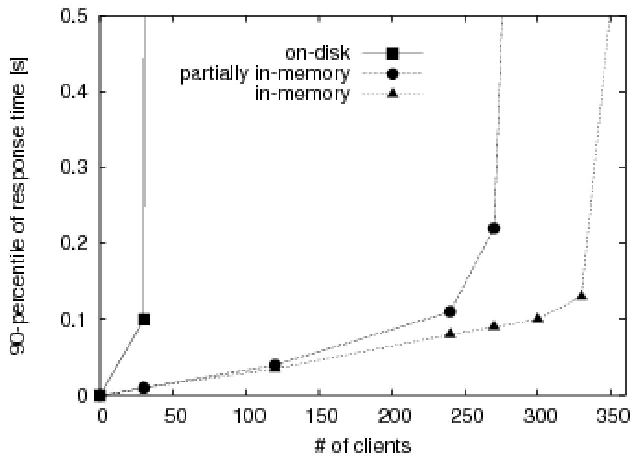


Figure 3. Response time of the Web system for different memory scenarios

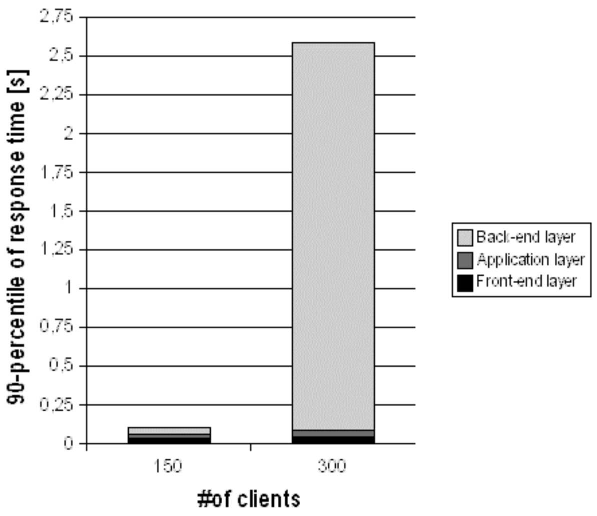


Figure 4. Breakdown of the Web system response time (*partially in-memory scenario*)

and number of open sockets. For this reason, we have instrumented a system monitor that provides regular samples of these three performance indexes on every node of the Web-based cluster infrastructure. As we will see, our analysis shows that these three indexes allow us to identify the resource bottleneck under every considered memory scenario, and to evidence how the system bottlenecks change depending on the amount of main memory.

On-disk scenario

The most significant results about the white-box analysis for the on-disk scenario are shown in Figure 5 that reports the main performance parameters of the back-end node as a function of the time elapsed from the beginning of the experiment (*x*-axis).

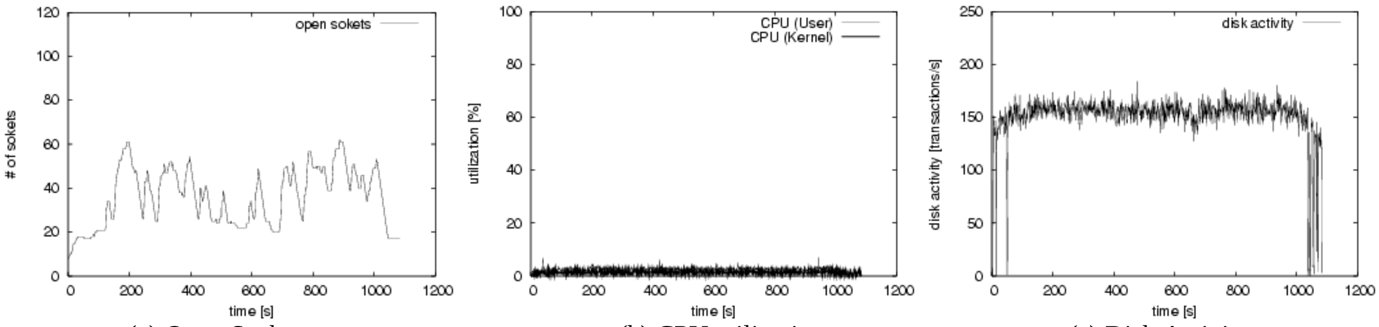
In particular, Figure 5(a) shows the number of open sockets at the back-end server node throughout 1000 seconds of the experiment. Figure 5(b) shows the CPU utilization that is split into kernel and user mode: the kernel mode refers to the CPU operations related to the operating system, such as process scheduling, context switches and system call services; the user mode relates to the application processes. Since we are monitoring the back-end node, the CPU utilization in user mode is almost exclusively due to DBMS operations. Finally, Figure 5(c) shows the disk I/O activity that is reported as disk transactions per second throughout the experiment.

The first observation deriving from Figure 5(b) is a very low CPU utilization (below 0.1). This allows us to exclude the CPU from the list of possible bottleneck resources. Next, we observe a number of simultaneously open sockets that is significantly lower than in other experiments (see Figure 6(a)), but that is highly variable with a range spanning from 20 to 60 open sockets. This leads us to conclude that the number of open sockets does not represent a system bottleneck for this scenario because 60 is far below the number of 105 sockets that are available at the database server.

However, the high variability of the number of open sockets deserves some motivations. Since the database server is equipped with a limited amount of memory, most database operations are delayed by the disk accesses. The long database service time increases the number of concurrent client requests, thus leading to a high number of open sockets even for low numbers of user requests.

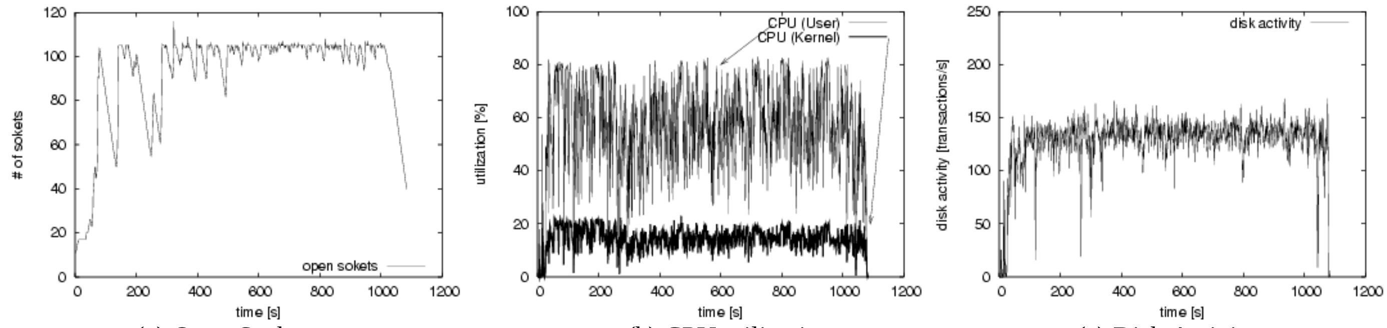
When we pass to analyze the disk performance, we notice an almost constant throughput in terms of operations, always close to 150 transactions per second. To have a guarantee that the disk represents the system bottleneck for the on-disk scenario, we evaluated the maximum disk throughput for the considered architecture. To this purpose, we use the *iozone* [15] tool which allows us to evaluate the maximum throughput when the disk is subject to complex read/write/seek patterns, as in the case of database operations of the TPC-W workload that we have used for the experiments.

Other related tests show that 160 disk operations per second represent the maximum achievable throughput for the considered workload. This result evidences that the disk is the only system resource that operates close to the maximum of its capacity.



(a) Open Sockets (b) CPU utilization (c) Disk Activity

Figure 5. Resource utilization on back-end node (*on-disk scenario*)



(a) Open Sockets (b) CPU utilization (c) Disk Activity

Figure 6. Resource utilization on back-end node (*partially in-memory scenario*)

Partially in-memory scenario

Figures 6(a), (b) and (c) show the socket, CPU and disk utilization for the partially in-memory scenario, respectively. With respect to the on-disk scenario, in this case, we observe a significant increment of the CPU utilization and the number of contemporary open sockets, while the disk activities tend to decrease. It is worth to consider that these utilization levels are reached for a number of served requests that is more than nine times the number related to the on-disk scenario (from 30 to 270). This consideration motivates the sudden increment of the CPU utilization combined with a low decrement of the disk utilization, even although more than half of the database is kept in main memory. But the most significant result is that for the partially in-memory scenario, neither the CPU nor the disk of the database server are system bottlenecks. Initially unexpected, the system capacity in this case is limited by the number of sockets. Indeed, from Figure 6(a) we can observe that the number of open sockets on the DBMS node is always equal to the maximum capacity that is set to 105.

In-memory scenario

Figure 7 shows the resource utilization for the in-memory scenario. Identifying the system bottleneck for this case is a straightforward task, because the disk is almost not used at all (Figure 7(c)); the number of open sockets is around 20 (see Figure 7(a)), a value that is far below the maximum limit set to 105. On the other hand, Figure 7(b) shows a CPU utilization close to 1 with a 0.8/0.2 ratio between the time spent in user and kernel space. The immediate conclusion is that for the in-memory scenario the system bottleneck is represented by the CPU of the back-end node.

This initially unexpected result suggests that the computations at the application level are much more intensive than the cost necessary for the system calls. As there is only one major process running on the back-end node, we can easily assume that the DBMS process is the real source of the system bottleneck.

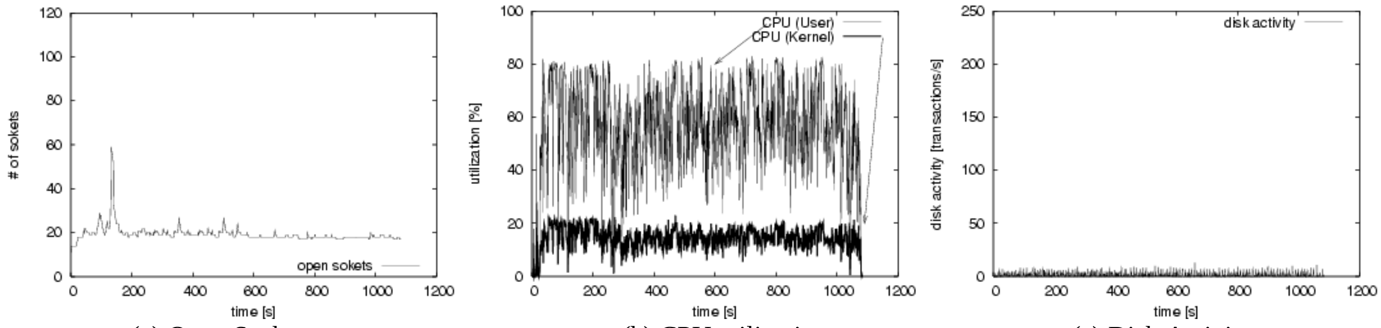
Nevertheless, if we limit the performance analysis at this level of granularity, we cannot exactly motivate the high CPU utilization of the database server application. To identify the hot spots in the database server process we should pass to get measurements at the function level. To this purpose, we utilize a system profiler from which we get that the function that checksums asynchronous I/O buffers uses almost 70% of the CPU time. This function is part of the asynchronous I/O buffer management of the MySQL DBMS. Asynchronous I/O is used to improve I/O performance by caching frequently accessed portions of the database, thus bypassing the operating system disk buffer cache. To provide data consistency a checksum is calculated on every buffer. Hence, we can conclude that the asynchronous I/O subsystem is the real bottleneck of the mysqld process.

It is tricky to solve this CPU bottleneck at the database server. The most straightforward option seems to purchase a faster CPU that can provide higher computational power. However, this solution has a limited scalability. The best alternative to improve the database performance is to reduce the checksumming activity by decreasing the number of buffer accesses. For example, this can be easily carried out by augmenting the size of the query cache.

Summary of results

We can conclude that the amount of memory on the back-end node of a multi-tiered Web system is having and will have a fundamental impact on system performance. Although some results were expected, the experiments evidenced some novelties. For example, increasing the available memory augments the system capacity, but does not reduce significantly the response time observed by the client. Moreover, it is quite interesting to verify how the memory availability alters in a fundamental way the system bottleneck that limits the performance of the architecture.

In particular, the possibility of storing half of the Web site information in main memory has a super-linear benefit on the system capacity that in our case improves of more than eight times. Storing even the remaining part of the Web site information in memory augments the system capacity of "just" an additional 30%. With different system architectures, the effects would be different, even if our experience leads us to conclude that the first half memory storage produces the main increment of the system capacity. The in-memory scenario does not improve the system capacity with respect to



(a) Open Sockets (b) CPU utilization (c) Disk Activity

Figure 7. Resource utilization on back-end node (in-memory scenario) the partially in-memory scenario such as it has been done in the passage from the in-disk to the partially in-memory scenario. Indeed, the system for the partially in-memory scenario works for high request rates, but it is in very bad conditions that is, the disk and the CPU are really highly utilized, and the set of available sockets is almost exhausted. A platform where all the most important system resources are critically loaded is something that any system manager must avoid. In some sense, this situation is even worse than that shown by the on-disk and of in-memory scenarios where just the disk (or the CPU) is a clear system bottleneck in a context where the other resources are not critically loaded.

VII. IMPACT OF NETWORK IMPROVEMENTS

We now evaluate the impact of the trends of the network technology on the system performance and related effects on the system bottlenecks. Many parts of the Internet, including the last mile links, are being characterized by larger bandwidth, lower latency and lower packet losses, and this trend is likely to increase.

For evaluating the consequences that this trend may have on Web-based services, we use the same testbed architecture and workload model of the previous section, with the addition of an emulator of WAN effects between the clients and the front-end node of the Web system. We consider the all in-memory database and partially in-memory database scenarios, and two main network scenarios: *well connected* and *badly connected* clients. The experiments related to these scenarios lead to the four curves reported in Figure 8 where we show the results of a black-box testing. This figure reports the 90-percentiles of the response times as a function of the number of requests reaching the Web site (i.e., client population).

As expected, the quality of the connection has a direct impact on the overall response time. Passing from a large majority of clients that are badly connected to the opposite scenario has a twofold effect on performance: good network connectivity reduces the response time by an order of magnitude, and the Web system can serve a larger number of requests.

It is interesting to observe that the network connectivity seems to have an impact on performance even more consistent than the possibility of having a memory-embedded Web site. The two couples of curve for badly connected clients are close, although the improvement of the all in-memory scenario is the most sensible. The system capacity for the partially in-memory scenario in terms of managed requests increases by 50% (from 180 to 270 clients), while for the all in-memory scenario the increment reaches 83% (from 180 to 330 clients). However, the real impact of a combined trend that improves network connectivity and augments the main memory size cannot be fully explained without a white-box performance analysis that aims to identify the system bottlenecks.

For the partially in-memory scenario, the resource utilization is not significantly modified by the characteristics of the network connections. The number of open sockets on the database server for both network scenarios remains the system bottleneck. Hence, we do not report the results for this memory scenario.

If we focus on the all in-memory scenario, we can observe a clear modification of the system bottleneck depending whether we consider good or badly connected clients. In the latter case, the disk activity is low, and the bottleneck is related to the number of open sockets. In the former case, the bottleneck remains the CPU of the database server.

Figure 9 shows the number of open sockets sampled during the experiment for both network scenarios. From this figure, it is immediate to get two results. First, the number of sockets simultaneously used by the database server is five times higher in the badly connected scenario with respect to the good connection case: on average, 20 vs. 105, where 105 is the maximum number of concurrent connections for the database. Second, in the badly connected scenario, all available socket descriptors at the database are used. The

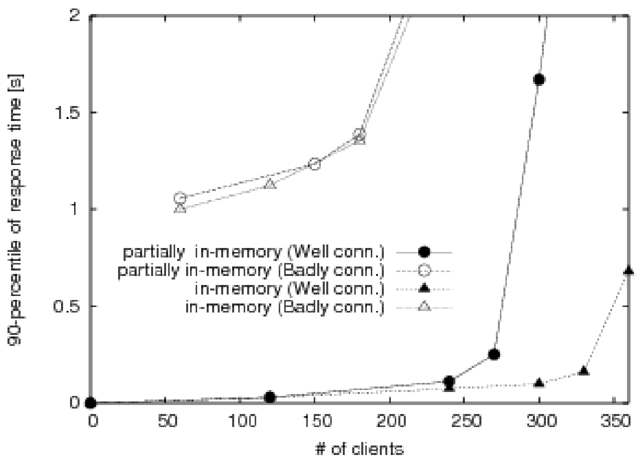


Figure 8. Response time for different types of network connections and main memory size

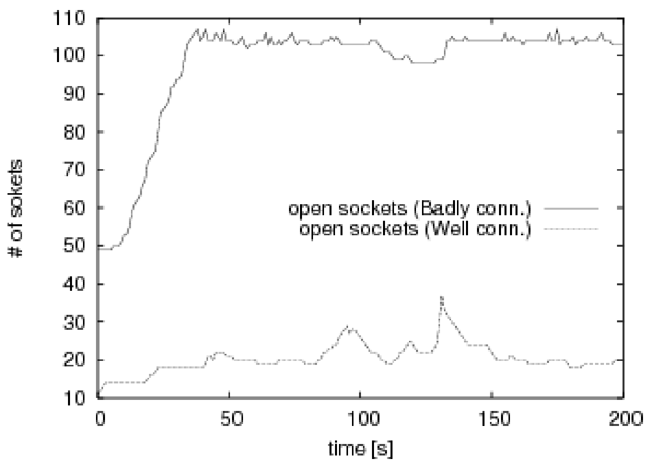


Figure 9. Number of utilized sockets by the back-end server in the two network scenarios

exhaustion of the number of open sockets due to poor network connectivity is a phaenomenon that resembles the socket bottleneck caused by many delayed disk accesses. As for the delay introduced by the disk access, the connections between the application server and the database server last much longer in the badly connected scenario because of the network slowdown of client requests. We should consider that sockets are token-based resources that are not gracefully degradable. This means that, once the number of available sockets is exhausted, further requests to the database server must wait without receiving service until a token is freed. Hence, the contention due to accessing the limited pool of available sockets results in an additional queuing delay. This delay (waiting for a free socket) has the effect of further increasing the concurrency level of requests on the database. This leads to an amplification of the phenomenon that resembles thrashing. The macroscopic effect of socket shortage is the poor performance of the application server and back-end server components.

The insight provided by the bottleneck analysis allows us to explain the similarity between the two performance curves shown in Figure 8 in the badly connected scenario for the considered memory cases. When the wide area network effects introduce a significant performance penalty, both the all in-memory and the partially in-memory scenarios are characterized by the same bottleneck due to socket exhaustion.

Table 2 summarizes the bottlenecks for the four combinations of memory and network scenarios. It is worth to note that the bottlenecks due to socket exhaustion are likely to arise whenever a client request service is delayed, either by the disk or by the network. An important consequence, that we should consider if we want to improve the system capacity, is that increasing the CPU power and/or augmenting the main memory size for storing the entire database do not solve every performance problem. Indeed, a special attention on token-based resources is highly recommended especially because they are likely to be consumed in the case of high concurrent load. This may lead to poor performance even in over-provisioned systems with large amount of RAM and computational power.

	PARTIAL IN-MEMORY	All In-memory
Well connected clients	socket	CPU
Badly connected clients	socket	socket

Table 2. System bottlenecks for the conidered memory and network scenarios

VIII. Related work

The literature regarding the impact of technological trends on the performance of devices and services is scarce. One of the few examples is provided in [24], where the author discusses the state of the art of the late 1990's CPUs and the latest technological trends of chip development. The work is interesting because it contains a precise estimate of how the CPU work frequencies are supposed to increase until 2012.

Multiple papers compare different technologies for dynamic Web-based sites. For example, in [1, 10, 12, 17, 26] the authors evaluate the performance of J2EE and PHP implementations of the same e-commerce system. However, the performance comparison in these studies is limited to the scalability of each technology and does not focus on the impact of technological trends on the system bottlenecks, as carried out by our paper. Furthermore, all these studies share the common trait of focusing on a coarse grain performance analysis of the systems at most at the node level, with no evidence of the real causes behind poor performance. Conclusions are inferred either from indirect measurement or from coarse-grained system activity reports. On the other hand, our performance evaluation integrates both a coarse and a fine grain analysis to provide a deep insight on the causes of the system bottlenecks.

Other studies illustrate different aspects of Web systems testing. For example, in [27, 14] a fine grained analysis of the performance on HTTP servers is provided. However, these studies focus on Web sites serving mainly static Web resources and do not take into account the complexity and the interactions of a Web site providing highly dynamic and personalized contents.

There is no general consensus on the most useful performance indexes for the evaluation of a dynamic Web-based system, although some effort has been put on their identification. For example, in [3] the authors study the performance of dispatching algorithms in multi-tier architectures, providing some insights into the choice of appropriate load monitoring indexes. They find out that the most critical load monitoring index is that associated to the bottleneck resource of the system. M. Dahlin [9] addresses the problem of using stale server load information in the context of distributed systems. The author proposes some algorithms for interpreting server state information based on its age that may help to improve the performance of dynamic Web-based systems.

The effects of Wide Area Networks on Web server performance has been pointed out in [21, 5]. However, both studies are limited to a single Web server hosting static contents, and do not take into account the consequences of different client bandwidths on the performance of the entire Web system. This is a limit because our experiments have demonstrated that packet delays and losses have a great influence even on the performance of the back-end server.

IX. Conclusions

There is a close relationship among the bottlenecks limiting the system performance and the hardware characteristics of the underlying platform. Hence, to achieve adequate performance improvement it is important to investigate the technology trends so to anticipate the types of interventions that should be undertaken at the operating system and server software level.

Our study is a first step towards this direction and gives some clear messages about the impact that current technological improvements concerning main memory and network capacity may have on performance and bottlenecks of Web-based services. In particular, we consider an e-commerce Web site implemented on a multi-tier architecture, that is subject to a TCP-W like workload model. Throughout a large set of experiments for different memory and network scenarios, we confirm some intuitions and achieve other less expected results. In particular, we show how the bottlenecks limiting system performance change depending on the amount of main memory available at the server machines and on the characteristics of the network interconnection between the client and the Web system.

This paper has shown that a clear understanding of technological trends and an analysis of their implications on Web systems help to foresee both present and future bottlenecks that hinder the performance of the Web-based services. Such a knowledge is also invaluable when it comes to choose the main interventions during system consolidation and capacity planning studies.

References

[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *Proc. of the IEEE 5th Annual Workshop on Workload Characterization (WWC-5),* Nov 2002.

[2] M. Andreolini, M. Colajanni, R. Lancellotti, and F. Mazzoni. Fine grain performance evaluation of e-commerce sites. *ACM Performance Evaluation Review*, 32(3), Dec. 2004. Special Issue on E-Commerce.

[3] M. Andreolini, M. Colajanni, R. Morselli. Performance study of dispatching algorithms in multi-tier Web architectures. In *ACM SIGMETRICS Performance Evaluation Review*, 30(2), pages 10-20, Sep 2002.

[4] Apache foundation, 2005, Apache httpd server - http://httpd.apache.org

[5] P. Barford, M. Crovella. Measuring Web Performance in the Wide Area. In *IMA "Hot Topics" Workshop: Scaling Phaenomena in Communications Networks*, Minneapolis, Oct 1999.

[6] V. Cardellini, E. Casalicchio, M Colajanni, P. S. Yu. The state of the art in locally distributed Web server systems. *ACM Computing survey* 34(2):263-311, 2002

[7] E. Cecchet, A. Chandra, S. Elnikety, J. Marguerite and W. Zwanapoel. Performance comparison of middleware architectures for generating dynamic Web content. In *Proc. of 4th middleware conference*, Jun. 2003

[8] W. Chiu Design pages for performance, IBM High Volume Web Systems, 2001

[9] M. Dahlin. Interpreting Stale Load Information. In *IEEE Transactions on Parallel and Distributed Systems*, 11(10), Oct 2001.

[10] R. C. Dodge, D. A. Menasce, and D. Barbara. Testing e-commerce site scalability with TPC-W. In *Proc. Of 2001 Computer Measurement Group Conference*, Dec. 2001.

[11] S. Godard. Sysstat: System performance tools for linux OS, 2004. – http://perso.wanadoo.fr/sebastien.godard/.

[12] X. He and Q. Yang. Performance evaluation of distributed web server architectures under e-commerce workloads. In *Proc. of the 1st Int'l Conference on Internet Computing (IC'2000)*, Jun 2000.

[13] S. Hemminger netem: Network emulator, 2005. – http://developer.osdl.org/shemminger/netem/.

[14] Y. Hu, A. Nanda, and Q. Yang. Measurement, analysis and performance improvement of the apache web server. *International Journal of Computers and Their Applications*, 8(4), Dec. 2001.

[15] IOzone filesystem benchmark, 2005. – http://www.iozone.org/.

[16] Java Technology. Java 2 platform,enterprise edition (J2EE), 2005 – http://java.sun.com/j2ee/index.jsp.

[17] K. S. Juse, S. Kounev, and A. Buchmann. Petstore-ws: Measuring the performance implications of web services. In *Proc. of the 29th Int'l Conference of the Computer Measurement Group (CMG) on Resource Management and Performance Evaluation of Enterprise Computing Systems -CMG2003*, Dec. 2003.

[18] G. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[19] D. Morse. In memory database Web server. *Dedicated systems magazine*, 2000.

[20] MySQL database server, 2005 – http://www.mysql.com/

[21] E. M. Nahum, M.-C. Rosu, S. Seshan, and J. Almeida. The effects of wide-area conditions on www server performance. In *Proc. of the 2001 ACM SIGMETRICS int'l conference on Measurement and modelling of computer systems*, pages 257– 267, 2001.

[22] PHP scripting language, 2005 – http://www.php.net/.

[23] M. Rabinovich and O. Spatscheck. *Web caching and Replication*, Addison Wesley, 2002

[24] U. Rude. Technological trends and their impact on the future of supercomputers. In *International FORTWIHR Conference on HPSEC*, Muenchen, Mar 1998.

[25] Standard Performance Evaluation Corporation (SPEC). SPECweb2005 suite - http://www.spec.org/benchmarks.html#web

[26] L. Titchkosky, M. Arlitt, C. L. Williamson. A performance comparison of dynamic Web technologies. In *SIGMETRICS Performance Evaluation Review*, 31(3), pages 2-11, Dec 2003.

[27] H. Xie, L. Bhuyan, and Y.-K. Chang. Benchmarking web server architectures: A simulation study on micro performance. In *Fifth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-02), with HPCA-8*, Feb 2002.

Mauro Andreolini is currently a researcher in the Department of Information Engineering at the University of Modena, Italy. He received his master degree (summa cum laude) at the Univeristy of Roma, "Tor Vergata", in January 2001, and the Ph.D. degree in computer engineering from the University of Roma "Tor Vergata" in 2004. In 2003, he spent eight months at the IBM T.J. Watson Research Center as a visiting researcher.
His research focuses on the design, implementation and evaluation of distributed Web server systems, based on a best-effort service or on guaranteed levels of performance. He is a Standard Performance Evaluation Corporation (SPEC) technical responsible for the University of Modena. He has been in the organization committee of the IFIP WG7.3 International Symposium on Computer Performance Modeling, Measurement and Evaluation (Performance2002). For additional details, see: http://weblab.ing.unimo.it/people/andreoli

Michele Colajanni is a full professor of computer engineering at the Department of Information Engineering of the University of Modena. He was formerly an Associate Professor at the same University in the period 1998-2000, and a Researcher at the University of Roma Tor Vergata. He received the Laurea degree in computer science from the University of Pisa in 1987, and the Ph.D. degree in computer engineering from the University of Roma "Tor Vergata" in 1991. He has held computer science research appointments with the National Research Council (CNR), visiting scientist appointments with the IBM T.J. Watson Research Center, Yorktown Heights, New York. In 1997 he was awarded by the National Research Council for the results of his research activities on high performance Web systems during his sabbatical year spent at the IBM T.J. Watson Research Center.
His research interests include scalable Web systems and infrastructures, parallel and distributed systems, performance analysis, benchmarking and simulation. In these fields he has published more than 100 papers in international journals, book chapters and conference proceedings. He has lectured in national and international seminars and conferences.
Michele Colajanni has served as a member of organizing or program committees of national and international conferences on system modeling, performance analysis, parallel computing, and Web-based systems. He is the general chair of the first edition of the International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA2005). He is a

member of the IEEE Computer Society and the ACM. For additional details, see: http://weblab.ing.unimo.it/people/colajanni

Riccardo Lancellotti received the Laurea and the Ph.D. degrees in computer engineering from the University of Modena and from the University of Roma “Tor Vergata”, respectively. He is currently a Researcher 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 research student.
His research interests include scalable architectures for Web content delivery and adaptation, peer-to-peer systems, distributed systems and performance evaluation. Dr. Lancellotti is a member of the IEEE Computer Society. For additional details, see: http://weblab.ing.unimo.it/people/riccardo