2 Web system reliability and performance: Design and Testing methodologies

Mauro Andreolini, Michele Colajanni, Riccardo Lancellotti

Abstract. Modern Web sites provide multiple services that are deployed through complex technologies. The importance and the economic impact of consumer-oriented Web sites introduce significant requirements in terms of performance and reliability. This chapter presents some methods for the design of novel Web sites and for the improvement of existing systems that must satisfy some performance requirements even in the case of unpredictable load variations. The chapter is concluded with a case study that describes the application of the proposed methods to a typical consumer-oriented Web site.

Keywords: Web systems, Performance, Reliability, Design

2.1 Introduction

The Web technologies are becoming a de facto standard for the deployment of many classes of services through the Internet. A huge number of software and hardware technologies are available with their pros and cons in terms of complexity, performance and cost. Because of the extreme heterogeneity of Web-related services and technologies, it is impossible to identify from this universe the solution which suits best to every possible Web site. Multiple issues should be addressed in the design and deployment of a Web-based service, including efficacy of the presentation, richness of the provided services, guarantee of security. Moreover, system performance and reliability remain key factors for the success of any Web-based service. The popularity of a Web site, that is perceived as too slow or that suffers availability problems, drops dramatically because navigation becomes a frustrating experience for the user. There are specific aspects of the design process that focus mainly on data organization. The interested reader can refer to [24] and to references therein. In this chapter, we focus instead on the architectural design of Web sites that are subject to some performance and reliability requirements. It is worth to premise that we remain in the domain of best-effort Web-based services, with no strict guarantees on the performance levels, similarly to those that characterize QoS-based applications [44, 23]. The techniques described in this chapter are hybrid in nature, but there are some major steps that must be followed. These main procedural steps are illustrated in Figure 2.1 and detailed in the following sections.

1.We have first to identify the main classes of services that must be provided by the Web site.
2.As we are interested to serve a large number of users with many *classes* of services, it is important to define the most likely workload models that are expected to access the Web site. Each workload model is characterized by a *workload intensity* which represents the number of requests admitted in the Web site and by the *workload mixes*, that denote the number of requests in each class of service.
3.The third step in the design phase is to define the performance and reliability requirements of the Web site. There are many system- and user-oriented performance parameters as well as many types of reliability goals, the most severe of which is the well known 24/7 attribute (i.e., 24 hours, seven days a week) that aims to deploy always reachable Web sites.
4.Once the main characteristics and requirements for the Web site are specified, we have to choose the most appropriate software and hardware technologies to deploy it.
5.After the implementation phase, we have to verify whether the Web site works as expected and whether it respects the performance and reliability requirements. As usually, a test can lead to positive or negative outcomes.
6.In the case of some negative results, an iterative step begins. It aims to understand the causes of violation, to remove them, to check again until all expected performance requirements are satisfied. In the most severe cases, a negative outcome may require interventions at the system or implementation level (dashed line in Figure 2.1)
7.Often, even the positive conclusion of the tests does not conclude the work. If one considers the extreme variability of the user patterns, the frequent updates/improvements of the classes of services, the first deployment may be followed by a *consolidation* phase. It can have different goals, from capacity planning tests to the verification of the performance margins of the system resources.

Section 2.2 outlines the different types of Web sites and the main design challenges for each class. Section 2.3 describes software technologies and hardware architectures for the deployment of Web sites that must serve mainly dynamic Web resources. Section 2.4 focuses on the testing process of a Web site. Section 2.5 outlines the main countermeasures to be undertaken whenever the deployed Web system fails to meet performance and reliability requirements. Finally, Section 2.6 describes a case study showing how the proposed design and testing methods can be applied to a typical e-commerce Web site of medium size. Section 2.7 concludes the chapter with some final remarks.

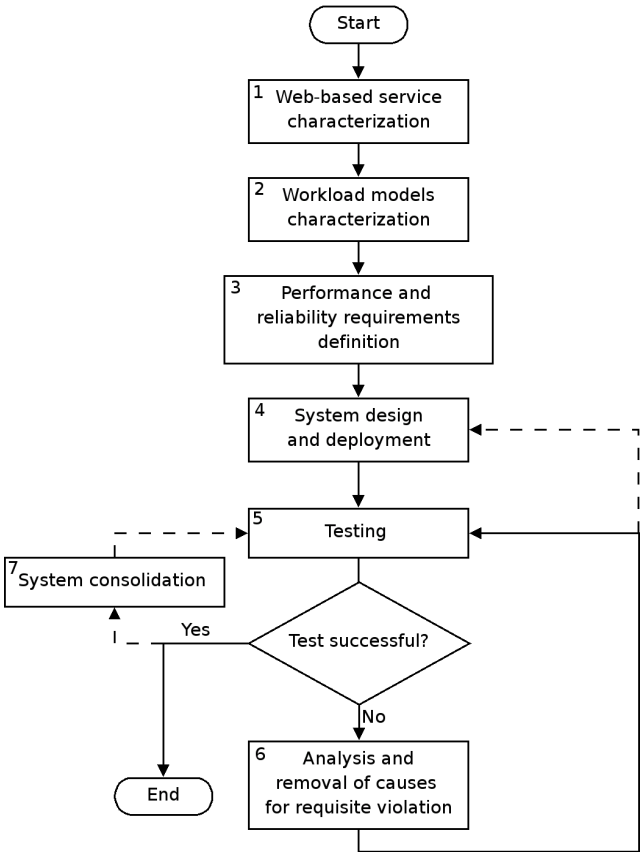

Fig. 2.1 Procedural steps for designing and testing Web sites with some performance requirements

2.2 Web site services

There are so many services provided through the Web that it is difficult to integrate all of them in a universally accepted taxonomy. We prefer to describe the Web-based services through the following considerations:

- Each class of requests to the Web site involves one or more types of *Web resources*, hence we consider that the first important step is to classify the most important resources that characterize a site.
- Each class of requests has a different impact on the *system resources* of the platform that supports the Web site. The system resources include hardware and software components that are typically based on distributed technologies.

Knowing the available technologies and their interactions is fundamental for the design of performance-aware Web sites.

2.2.1 Web resource classification

We recognize five basic resource types that are provided by a Web site. Servicing each of these Web resources requires specific technologies and has a different computational impact on the system platform.

Static Web resources are stored as files. There are dozen of static resource types, from HTML files to images, text files, archive files, etc. They are typically managed by the file system of the same machine that hosts the Web server. In the first years of the Web, static files were the only type of Web resources. Servicing them does not require a significant effort from the Web system, since a static resource is requested through a GET method of the HTTP protocol, then fetched from a storage area or, often, from the disk cache, and then sent to the client through the network interface. Some performance problems may occur only when the static resource is very large (with present technology, from some Mbytes upwards).

Dynamic Web resources are generated on-the-fly by the Web system as a response to some client request. There are many examples of dynamic resources, from the result page of some Web search, to the shopping cart virtualization in a Web store, to the dynamic generation of some embedded objects or frames. Web resources generated in a dynamic way allow the highest flexibility and personalization because the page code is generated as a response to each client request. There are two main motivations behind the use of dynamic resources.

The traditional dynamic request comes from the necessity of getting some answers from an organized source of information, such as a database. The generation of this type of responses requires a significant computational effort due to data retrieval from databases, (optional) information processing and construction of the HTTP output. The computational impact of dynamic requests on a Web site is increased also by the fact that it is quite difficult to take advantage of caching for the dynamic resources.

One of the new trends in the Web is the generation of dynamic content even when this is not strictly necessary. For example, XML- and component-based technologies [19, 29] provide mechanisms for separating structure and representation details of a Web document. As a consequence, all the documents (even static pages) are generated dynamically from a template through computationally expensive operations.

Volatile Web resources are (re-)generated dynamically on a periodical time basis or when some events occur. These types of resources characterize information portals that deliver up-to-date news, sport results, stock exchanges, etc. Avoiding frequent re-computation keeps the cost of volatile resource service low and comparable to that of static resources. On the other hand, the Web system must be equipped with some mechanisms that re-generate resources through automated technologies similar to those used for dynamic Web resources [40, 21]. Some pushing methods to the clients are sometime utilized [41].

Secure Web resources are static, dynamic or volatile objects that are transferred over a ciphered channel, usually through the HTTPS protocol. Secure resources address the need for privacy, non-repudiation, integrity, and authentication. They are

typically characterized by high CPU processing demands, due to the computational requirements of the cryptographic algorithms [18, 25].

Multimedia resources are associated with video and audio content, such as video clips, mp3 audio files, Shockwave Flash animations, movies. There are two main ways for enjoying multimedia resources: download-and-play, or play-during-download. In the former case, multimedia content is usually transferred through the HTTP protocol. The typical size of a multimedia resource is much larger than that of other resources, hence the download traffic caused by these files has a deep impact on network bandwidth requirements. In the case of play-during-download, the download service must be integrated with some streaming-oriented protocols, such as RTP [21, 43], and some well designed technologies for providing content from the Web site with no interruption.

2.2.2 Web site impacts on system resources

Over the years, the Web has evolved from a hostel of simple, immutable home pages to the main interface for sophisticated dynamic and secure services, such as these behind e-commerce and home banking sites. A complete taxonomy that takes into account every Web site is virtually impossible. To the purpose of a design that is oriented to consider performance and reliability, it is important to consider the four main hardware resources of a system platform that is, CPU, disk, central memory, network interface. Moreover, it is more important to focus on the classes of requests than on the types of resources offered by a Web site. Indeed, each site consists of a mix of Web resources, but for each workload mix there is a prevalent class of requests that has the major impact on the system resources. As an example, if the workload model is prevalently characterized by downloads of multimedia files, the network capacity has a primary impact on system performance; hence it is important to design an architecture that is able of guaranteeing high network throughput. For these reasons, we classify some common Web sites according to their prevalent request class.

Prevalent static sites. Today, static Web sites do not present any real design challenge, because present Web technologies are able to serve an impressive volume of static requests even with commodity off-the-shelf hardware and software. The only requirement that a static Web site has to meet concerns the network capacity of the outbound link, which must handle the necessary volume of client requests/responses with no risk of bottleneck.

Prevalent dynamic sites. Sites offering sophisticated and interactive services, possibly with personalized content, fall in the category of dynamic sites. A peculiarity of dynamic sites is the strong interaction between the Web technology and the information sources (usually, databases) for nearly every client request. To provide adequate performance for serving dynamic resources may be a non-trivial task because there are several technologies for dynamic content generation, each with some conveniences and limits. Choosing the wrong technology may lead to poor performance of the whole system. Section 2.3 is entirely dedicated to the analysis of dynamic Web sites and related technologies.

Prevalent secure sites. Secure Web sites provide services that are protected for security and privacy concerns. Significant examples include on-line shopping sites,

auctions sites, and home-banking services. Purchase is the most critical operation in secure e-commerce sites, because sensitive information (e.g., credit card number) is exchanged. When users buy, security requirements become significant and include privacy, non-repudiation, integrity, and authentication rules. The transactions should be tracked throughout the whole user session and backed up in the case of failures. The majority of the content of secure sites is often generated dynamically, however even static resources may need a secure transmission.

The highest computational requirement in secure sites is due to the initial public-key cryptographic challenge, which is needed to perform the authentication phase [18]. This is in accordance with a previous result [25], which confirms that the reuse of cached SSL session keys can significantly reduce client response time (from 15% to 50%) in secure Web-based services.

Prevalent multimedia sites. Multimedia Web sites are characterized by a large amount of multimedia content, such as audio and video clips, animations or slide-shows. Examples of multimedia sites include e-learning services, some e-commerce services specialized in music, such as iTunes [28], on-line radios, and sites that offer a download section with a repository of multimedia files.

We recall that two modes are available for multimedia resources fruition: file download or content streaming. In the former case, the primary design challenge is the same as for static Web sites: to provide enough network bandwidth for downloading large multimedia files. As multimedia resources are orders of magnitude larger than static resources, bandwidth requirements are quite critical. Introducing streaming protocols increases the issues in the design of a Web site because streaming-based delivery of multimedia contents introduces real-time constraints in packet scheduling [43], and often requires a network resource reservation protocol.

2.2.3 Workload models and performance requirements

Knowing the composition of each service in terms of Web resources gives a precise idea about the functional and software requirements for the design of the Web site, but a rough approximation for the design of the Web platform. Indeed service characterization alone does not permit to quantify the amount of system resources that are needed to meet the requested level of performance. For example, a service requiring many system resources, but characterized by infrequent accesses, may not have an impact on the Web system. On the other hand, another service with low resource requirements and frequent accesses may influence the performance of the entire Web system.

For these reasons, it is necessary to characterize a set of *workload models* that represent the behavior of clients when they access each service of the Web site. The combined knowledge deriving from both the service and the workload characterization permits to identify the system and Web resources that will be used most intensively. As we are interested to serve a large number of users with many classes of services, it is important to define the main workload models that are expected to access the Web site. Each workload model is characterized by the *workload intensity* that represents the typical number of requests admitted in the Web site, and by the *workload mixes* that are the number of requests in each class of service. Hence, we can consider expected workload models which reflect the typical volume and mix of requests that are supposed to reach the Web system, and worst

case workload models that reflect the maximum amount of client requests that are admitted in the Web site.

The next critical step is to quantify the level of adequate performance for the expected set of workload models. Only after this choice, it is possible to design and size the components and the system resources of the Web system according to some performance and reliability expectations. The problem here is that it is quite difficult to anticipate the possible offered loads to the Web site with no previous experience. The large number of system- and user-oriented performance parameters (some of which are reported in Section 2.4) as well as the types of reliability goals make even more complicated to define exact levels of adequate performance without testing the system under some representative workload models. In practice, the definition of the performance expectations is an iterative process by itself. During the design phase, the commissioner can give just rough ideas on workload intensity and mixes to the Web site designers and architects. It should be clear to both parts that the initial proposals do not represent a formal contract. On the other hand, the designers should be aware that it is preferable to choose a Web site architecture that guarantees a safe margin in expected performance (twofold as initially declared by the commissioner is not unusual).

Once the requirements of the Web system are defined in terms of Web resources, workload models and performance expectations, the design and deployment of the Web site become a matter of choosing the right software and hardware technology. To this purpose, it is important to know the main strengths and weaknesses of the most popular technologies. We review those related to the dynamic-oriented Web sites in the following section.

2.3 Sites with prevalent dynamic requests

For describing the hardware and software design of a Web site, we consider a system servicing a majority of dynamic resources. This type of site is highly popular and it introduces interesting design challenges, hence we consider it a representative case for describing the proposed methodology of design and testing.

2.3.1 Dynamic request service

Figure 2.2 presents an abstract view of the main steps for servicing a dynamic request. Three main entities are involved in the request service: the client, the Internet and the Web system. As we are more interested to the server part, we detail the Web system components. There are three main abstract functions that contribute to service a dynamic request: *HTTP interface*, *application logic* and *information source*.

The HTTP interface handles connection requests from the clients through the standard HTTP protocol and serves static content. It is not responsible for the generation of dynamic content. Instead, the functions offered by the *application logic* are at the heart of a dynamic Web system: they define the logic behind the generation of dynamic content and build the HTML documents that will be sent back to the clients. Usually, the construction of a Web page requires the retrieval of further data. The *information source* layer provides functions for storage of critical information

that is used in the management of dynamic requests that are passed to the application logic. The final result of the computations is an HTML (or XML) document that is sent back to the HTTP interface for the delivery to the client.

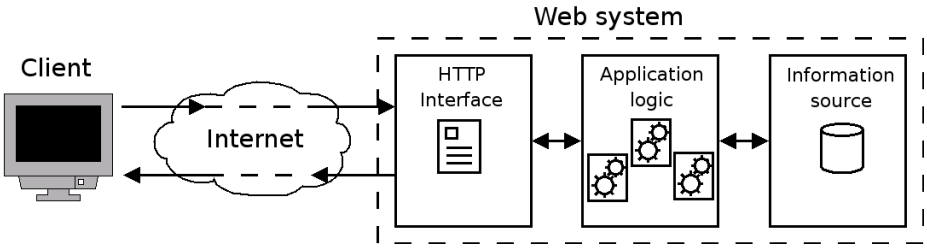

Fig. 2.2 Abstract view of a dynamic request

The use of three separate levels of functions has its advantages. The most obvious is the *modularity*: if the interfaces among different abstraction levels are kept consistent, changes at one level do not influence other levels. Another advantage is the *scalability*: the separation of abstraction layers makes it easier to deploy them on different nodes. It is even possible to deploy a single level over multiple, identical nodes. Section 2.3.3 provides some insights on these possibilities.

From Figure 2.2 we can get that the management of a dynamic request is the result of the interaction of multiple and complex functions. Each of them can be deployed through different software technologies that have their own strengths and weaknesses. Furthermore, they can be mapped in different ways to the underlying hardware. A performance-oriented design must address both issues: choose the right software technologies and the hardware architecture for the Web system. This is a non-trivial task that must be solved through an extensive analysis of the main alternatives at the software and hardware level.

From the point of view of software technologies, the real design challenge resides in the choice of the appropriate *application logic* because the HTTP interface and information source are well established. For the HTTP interface, Apache has become the most popular Web server [35], followed by other products, such as MS Internet Information Services, Sun Java System Web Server, Zeus. All of them provide the main functions of an HTTP server, but differ in the portability and efficiency levels that depend on operating systems and system platforms. Hence, the design of the HTTP interface becomes a simple choice among one of the aforementioned products that is compatible with the underlying platform and adopted software technologies. Apache works better with other open source products, such as PHP, Perl, Python. MS IIS works better with Microsoft software technologies.

Similar considerations hold for the information source layer, that handles the storage and retrieval of data. This layer consists of a database management system (DBMS) and some storage elements. There are many alternatives in the DBMS world, even if all of them are based on the relational architecture and some SQL dialect. The most common products are MySQL [33] and PostgreSQL [38] on the open source side, MS SQL Server, Oracle and IBM DB2 on the proprietary side. Hence, choosing the information source basically is a matter of cost, management, operating system constraints, internal competences, and taste.

In the following, we use a notation that is widely adopted in the Web literature. We refer to HTTP interface, application logic and information source also as *front-end layer*, *middle layer*, and *back-end layer*, respectively.

2.3.2 Software technologies for the application logic

The application logic of the middle layer is at the heart of a dynamic Web site. This layer computes the information which will be used to construct documents that are sent over a protocol handler. There is a plethora of software technologies which implement different standards. Each of them has its advantages and drawbacks with respect to performance, modularity, scalability. Let us distinguish the *scripting* from the *component-based* technologies.

2.3.2.1Scripting technologies

Scripting technologies are based on a language interpreter that is integrated in the Web server software. The interpreter processes the code that is embedded in the HTML pages and that typically accesses the database. The script code is substituted with its output, and the resulting HTML is returned to the client. Static HTML code (also called *HTML template*) is left unaltered. Examples of scripting technologies include language processors such as PHP [37], ASP [1] and ColdFusion [20].

Scripting technologies are efficient for dynamic content generation, because they are tightly coupled with the Web server. They are ideal for medium-sized, monolithic applications that require an efficient execution environment. Other applications that benefit from scripting technologies are characterized by large amounts of static, template HTML code that embeds a (relatively) small amount of dynamically generated data. An example is the ordinary product description page of an e-commerce site, which has a HTML template that is filled with variable information retrieved from the database.

On the other hand, the tight coupling between the front-end and the middle layer, which is typical of scripting languages, severely limits their use in Web-related applications that require high scalability. Indeed, to achieve scalability, it may be necessary to add nodes, but scripting technologies often lack integrated, high-level supports for coordination and synchronization of tasks running on different nodes. These supports can be implemented through the use of function libraries that are provided with the most popular scripting languages. However, this requires additional, significant programming efforts. For this reason, scripting technologies are seldom used to deploy highly distributed Web-based services.

2.3.2.2 Component-based technologies

Component-based technologies use software objects that implement the application logic. These objects are instantiated within special execution environments called *containers*. A popular component-based technology for dynamic Web resource generation is the Java 2 Enterprise Edition (J2EE) [29], which includes specifications for *Java Servlets*, *Java Server Pages* (JSP), and *Enterprise Java Beans* (EJB).

Java Servlets are Java classes that implement the application logic for a Web site. They are instantiated within a *Servlet container* (such as Tomcat [44]) that has an interface with the Web server. The object-oriented nature of Java Servlets enforces better modularity in the design, while the possibility to run distinct containers on different nodes facilitates a system scalability level that could not be achieved by the scripting technologies. Java Servlets represent the building block of the J2EE framework. Indeed, they only provide the low-level mechanisms for servicing dynamic requests. The programmer must take care of many details, such as coding the HTML document template, and organizing the communication with external information **sources**. For these reasons, Java Servlets are usually integrated with other J2EE technologies, such as JSP and EJB.

JSP is a standard extension defined on top of the Java Servlet API, that permits the embedding of Java code in an HTML document. Each JSP page is automatically converted into a Java Servlet that is used to serve future requests. JSP pages try to preserve the advantages of Java Servlets, without penalizing Web pages characterized by a large amount of static HTML template and a small amount of dynamically generated content. As a consequence, JSP is a better solution for dynamic content generation than plain Java Servlets, that are more suitable to data processing and client request handling. JSP is usually the default choice for dynamic, component-based content generation.

EJB are Java-based server-side software components that enable dynamic content generation. An EJB runs in a special environment called *EJB container*, that is analogous to a Java Servlet container. EJB provides native support for atomic transactions that are useful for preserving data consistency through commit and rollback mechanisms. Moreover, they handle persistent information across several requests. These added functions introduce a performance penalty due to their overhead. They should be used only in those services which require the user session persistence among different user requests to the same site. Common examples include database transactions and shopping cart services in e-commerce sites.

2.3.2.3 Technology comparison

An interesting performance comparison of scripting and component-based technologies is provided in [14]. This study compares the PHP scripting technology against Java Servlets and EJB for the implementation of a simple e-commerce site. Using the same hardware architecture, PHP provides better performance with respect to other component-based technologies. The performance gain is about 30% over Java Servlets, and more than double performance with respect to EJB. On the other hand, Java Servlets outperform the scripting technology when the system platform consists of a sufficient number of nodes.

Figure 2.3 shows a qualitative performance comparison of the two software technologies by considering the system throughput as a function of client traffic volume. From this figure we can see that scripting technologies tend to reach their maximum throughput before component-based technologies, because of their more efficient execution environment. Hence, component-based technologies tend to perform badly on small-to-medium sized Web sites, but they scale better than scripting technologies and can reach even higher throughput. The main motivation

lies in their high modularity, that allows to distribute the application logic among multiple nodes.

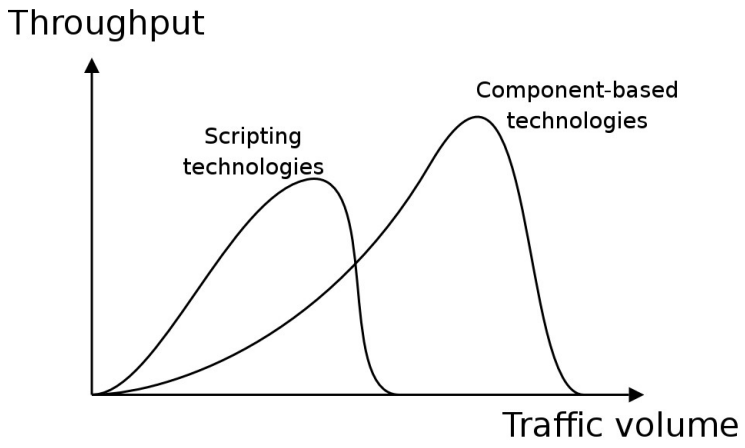

Fig. 2.3 A qualitative comparison of software technologies

2.3.3 System platforms

Once defined the logical layers and the proper software technologies that are needed to implement the Web site, we have to map them onto physical nodes. Typically, we do not have a one-to-one mapping because many logical layers may be located on the same physical node, as well as a single layer may be distributed among different nodes for the sake of performance, modularity and fault tolerance.

There are two approaches to map logical layers over the physical nodes, that we call *vertical* and *horizontal* replication. In a vertical replication, each logical layer is mapped to at most one physical node. Hence, each node hosts one or more logical layers. In horizontal replication, multiple replicas of the same layer are distributed across different nodes. Horizontal and vertical replication are usually combined together to reach a scalable and reliable platform.

The simplest possible hardware architecture consists of a *single node*, where all logical layers (front-end, middle, back-end) are placed on the same physical node. This architecture represents the cheapest alternative for providing a Web-based service; on the other hand, it suffers from multiple potential bottlenecks. In particular, the system resources can be easily exhausted by a high volume of client requests. Moreover, the lack of hardware component replication prevents the fault tolerance of a single node architecture. Explicit countermeasures such as RAID storage and hot-swappable redundant hardware may reduce the risks of single points of failure, but

basically there is no reliability opportunity. We should also consider that placing every logical layer on the same node has a detrimental effect on system security because once the node has been violated, the whole Web system is compromised. From the above mentioned considerations, we can conclude that the single node architecture is not a viable solution for the deployment of a dynamic Web site that intends to have some performance and reliability guarantees.

2.3.3.1 Vertical replication

In a vertical replication, the logical layers composing the Web-based service are placed into different nodes. The most common distributions for dynamic resource-oriented Web sites lead to the vertical architectures that are based on *two-nodes*, *three-nodes*, *four-nodes* schemes. Figure 2.4 shows the three examples of vertical replication.

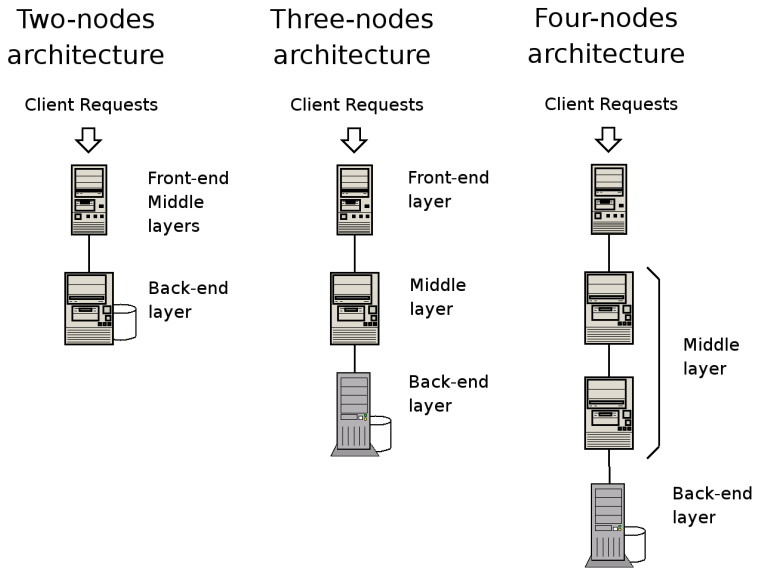

Fig. 2.4 Vertical replication

In the two-nodes architecture, the three logical layers are distributed over two physical nodes. There are three possible mappings between logical layers and physical nodes. However, the typical solution is to have the back-end layer on one node, and the front-end and middle layers on the other node. There are two main motivations for this choice. First, the tasks performed by a DBMS can easily exhaust the system resources of a single node. Second, front-end and application logic may be tightly coupled, as in the case of the scripting technologies; this makes separation of the logical layers very hard (if not impossible). The distribution over two nodes generally improves the performance of the Web system with respect to the single node architecture. Fault tolerance still remains a problem, because a failure in any of the two nodes causes a crash of the entire Web system.

In the three-nodes architecture, each logical layer is placed on a distinct node. Due to the tight coupling between front-end and middle layer in scripting technologies, the architecture based at least on three nodes is the best choice for component-based technologies. For example, the J2EE specification provides inter-layer communication mechanisms that facilitate the distribution of the front-end and the middle layer among the nodes. Scripting technologies do not natively have similar mechanisms, hence they have to be entirely implemented if the distribution of the layers over more than two nodes is a primary concern of the architectural design. Fault tolerance is still not guaranteed by three-nodes architectures, since a failure in any node hinders the generation of dynamic Web content. However, the three-nodes solution helps to improve the performance and reliability with respect to the two-nodes architecture, as shown in [14].

Four-nodes architectures are usually the choice to J2EE systems which distribute the middle layer among two physical nodes: one hosting the business logic which is encapsulated into the EJB container, and the others hosting the application functions through the JSP Servlet Engine. It is convenient to adopt this architecture because of the overheads of the EJB component.

Vertical replication is widely adopted not only for performance reasons. When security is a primary concern, this hardware architecture is useful because it allows the deployment of a secure platform through the use of firewalls between the nodes. The possibility of controlling and restricting communications among the nodes of the Web system aids in detecting security breaches and in reducing the consequences of a compromised system. In fact, the multi-layered architectures presented in Figure 2.4 are a simplification of the real systems that include network switches, authentication servers, and other security-oriented components.

2.3.3.2 Vertical and horizontal replication

Higher performance and reliability objectives motivate the tendency towards the replication of nodes at one or more logical layers, which is called *horizontal replication*. The latter is usually combined with vertical replication. Figure 2.5 shows the combination of horizontal and vertical replication. In particular, the figure shows a three-layer system where each of the three logical levels is hosted over a cluster of identical nodes, that are connected through a high speed LAN, each one running the same components. Initial distribution is achieved through a component called *Web switch*, which may be implemented in hardware or software. To achieve horizontal replication, other mechanisms are needed for distributing requests among the nodes of each layer.

When choosing the architecture of the Web system, it should be considered that the horizontal replication requires different efforts depending on the involved layer. For example, the replication of the front-end layer causes less problems because the HTTP protocol is stateless and different HTTP requests may be handled independently by different nodes [11].

The replication of the middle layer is rather complex. The motivation is twofold: the use of user session information by most applications and the type of software technology that is adopted for the implementation. Web-based services which make use of session information must be equipped with mechanisms that guarantee data consistency. Scripting technologies usually do not support these mechanisms

natively; some rely on external modules, others are forced to store session information in the back-end, with risks of serious slowdowns. Even in component-based technologies, the implementation of data consistency is not always immediate. For example, Java Servlets do not provide native persistent data support, while this is one of the strong attributes of EJB.

It is typically difficult to replicate horizontally even the back-end layer, because it introduces data consistency issues that must be addressed through difficult and onerous solutions [26]. Some modern DBMSes are equipped with the necessary mechanisms for guaranteeing horizontal replication of databases, but this replication is limited to few units.

The combination of vertical and horizontal replication helps improving important design objectives such as scalability and fault tolerance, which are crucial for obtaining an adequate level of performance and reliability. In particular, horizontal replication allows the use of dispatching algorithms that tend to distribute the load uniformly among the nodes [10, 3]. Moreover, hardware and software redundancy provided by horizontal replication also helps to add some fault tolerance to the system.

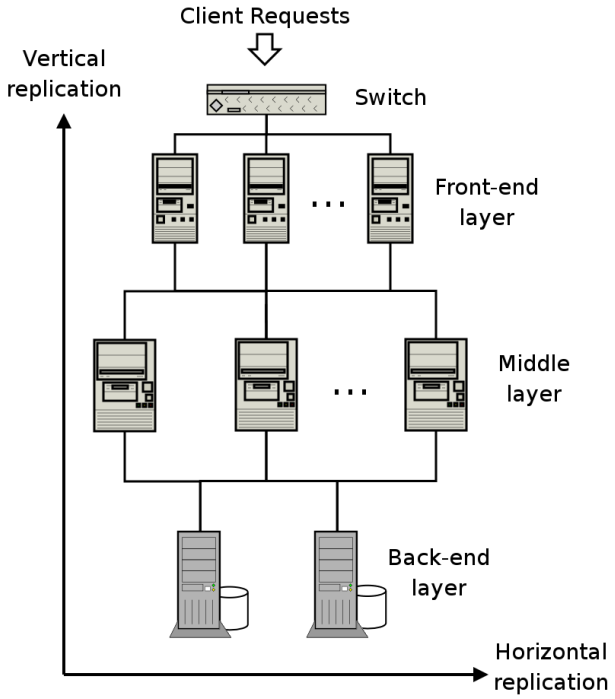

Fig. 2.5 Vertical and horizontal replication

A more complete fault tolerance requires also *fault detection* and *fail-over* mechanisms. Fault detection mechanisms monitor system components and check whether they are operative or not. When a faulty component is detected, the dispatchers may be instrumented to bypass that node; in the meanwhile, there are fail-over mechanisms that allow us to substitute the faulty node on the fly [12].

2.4 Testing loop phase

Once the Web-based service has been designed and deployed, it is necessary to verify the functional correctness of the services (*functional testing*) and the satisfaction of the performance and reliability requirements for each considered workload model (*performance testing*).

Functional testing aims at verifying that the Web site works as expected with no special interest on performance. This type of test is carried out by defining and reproducing typical behavioral user patterns for different operations. The output of each request is matched against an expected, desired template. Unexpected behaviors or errors imply that the Web system has not been deployed correctly, and that appropriate software corrections are needed. We will not delve into the details of Web system debugging that would require an entire chapter. We prefer to focus on performance testing, which allows us to verify whether the Web system guarantees the expected performance. The performance testing of a dynamic resource-oriented Web site is a non-trivial activity that requires the completion of different tasks: *representation of the workload model*, *traffic generation*, *data collection and analysis*.

2.4.1 Representation of the workload model

The first crucial step of testing is the translation of the workload models into a sequence of client requests that load generation tools can use to reproduce the appropriate volume of traffic. The choice of a request stream representation depends heavily on the complexity of the workload patterns. There are two main approaches, according to the complexity of the workload models.

Simple workload patterns, such as those of static resource-oriented Web sites, are usually well represented through *file lists* (with their access frequencies) and *analytical distributions*. While being fairly straightforward to implement, these two representations present drawbacks that limit their application to more sophisticated workloads. For example, file lists lack flexibility with respect to the workload specification, and do not provide any support for modeling the session-oriented nature of Web traffic. Analytical distributions allow us to define a wide characterization, being all features specified through mathematical models. It is an open issue to establish whether an analytical model reflects the user behavior in a realistic way. From our experience we can say that the large majority of studies has been focused on static content characterization [8, 9, 10], while still few studies consider Web sites with prevalent dynamic and personalized content. Some studies related to Web publishing sites can be found in [6, 45], the characterization of online shopping sites has been analyzed in [7, 45], preliminary results for trading and B2B sites can be found in [31] and [45], respectively.

The modeling of more complicated browsing patterns, such as those associated to on-line transactions, may require the creation of ad-hoc workloads, through the use of *file traces* and, in some case, the definition of *finite state automata*.

File traces of the workload model are based on pre-recorded (or synthetically generated) logs, derived from Web server access logs. Traces aim to capture the behavior of users in the most realistic way. On the other hand, the validity of tests depends strongly on the representativeness of a trace. Some of them may show characteristics peculiar to a specific Web site with no general validity. Furthermore, it may be hard to adjust the workload described by a trace to emulate future conditions or varying demands as well as to reconstruct user sessions.

The workload model may be described through a finite state automata, where each state is associated to a Web page. A transition from one state to another occurs with a predefined probability. A user think time is modeled as a variable delay between two consecutive state transitions. The main advantage of finite state automata lies in the possibility of defining complicated browsing patterns, which reflect modern consumer oriented Web sites. On the other hand, most of these patterns have to be specified manually, and this is an error prone operation.

2.4.2 Traffic generation

Once chosen the proper representation, the client request stream has to be replayed through a traffic generator. The main goal of a traffic generator is to reproduce the specified traffic in the most accurate and scalable way. Besides this, it also has to reproduce realistically the behavior of a fully featured Web browser, with supports for persistent HTTP connections, cookie management, or secure connections through the HTTPS protocol.

There are four main approaches to generate a stream of Web requests: *trace-based, file list-based, analytical distribution-driven* and finite state automata-based, depending on the workload model that is used as the base for traffic generation.

The reproduction of wide area network effects is another important factor which should be taken into serious consideration during the performance tests. It has been shown that, even in the presence of static resource-oriented Web sites, the performance evaluation is sensibly altered if the network is perturbed by routing delays, packet loss, client bandwidth limitation [34]. If these effects are not taken into consideration, typically because the Web site performance is evaluated in a LAN, the measured performance differs significantly from the reality, thus making the test results almost useless.

2.4.3 Data collection and analysis

Data collection is strictly related to the two main goals of the performance testing analysis: understanding whether a Web system is performing adequately and, if the expectations are not satisfied, finding the possible causes of performance slowdowns. They are the objectives of the so called *black-box* and *white-box* testing, respectively.

Data collection addresses two main issues of sampling: the choice of representative metrics for a given performance index, and the granularity of samples. The former problem is independent of the black and white-box testing. As Web workload is characterized by heavy-tailed distributions [9], many performance indexes may assume highly variable values with non negligible probability. Therefore, evaluating the performance only on the basis of mean, minimum and maximum values may not

yield a representative view of the Web system behavior. When performance indexes are subject to high variability, the use of higher moments, such as percentiles or cumulative distributions is highly recommended. This, in turn, typically requires the storage of every sample.

The choice of the sample granularity is related to the goal of testing. Sampling of performance indexes may occur at different levels of granularity, mainly *system* and *resource*. The former is more related to black-box testing, the latter to white-box testing.

2.4.3.1 Black-box testing

The main goal of black-box testing is to check whether the Web system is able to meet the performance and reliability requirements with safe margins for each workload model. The black-box test is related to *system performance indexes* that quantify the performance of the whole Web system, as seen from the outside. These are typically coarse-grained samples that aim to verify whether the Web system is performing adequately or not. Many performance indexes may be obtained from the running system for different purposes. For example, the throughput of the Web system in terms of served pages/hits/bytes per second may be of interest for the administrator, who wants the guarantee that the architecture is able to deliver the requested volume of traffic. The Web page response time that is, the time elapsed from the user click until the arrival of the last object composing a Web resource, is of main interest for the users that do not care on system throughput, but on the time they have to wait for the fruition of a given service. Both indexes reflect the performance of the entire Web system from different points of view. Although we are not considering a QoS-based Web site that requires the respect of rigorous Service Level Agreements (SLAs), it is convenient to refer to soft constraints that are generally accepted in the world of interactive Web-based services. For example, a previous study by IBM [16] provides a ranking of performance parameters (ranging from unacceptable to excellent) in terms of response time for a typical Web page loaded by a dial-up user. The study concludes that a Web page response time higher than 30 seconds is unacceptable, while everything below 20 seconds is considered at least adequate. In [36], the reaction of broadband users to different Web page download times is analyzed. One interesting conclusion is that the limit for keeping the user's attention focused on the browser window is about 10 seconds. Longer lasting Web page downloads lead the users towards other tasks.

We remark the importance that the Web system works within safe margins of performance. This claim has a twofold consequence: the system must respect the performance requirements for any expected workload; when the system is subject to the maximum expected workload intensity, it should not show signs of imminent congestion. The former requirement can be verified through a set of independent black-box tests for each representative workload model. The latter requirement is motivated by the observation that a Web system may meet all its performance requirements, but with some critically utilized resource. A similar situation is unacceptable because a burst of client arrivals may easily saturate the resource, thus slowing down the entire Web system. To avoid the risk of drawing false conclusions about the performance reliability of the system, we can pass to a white-box testing or

we can carry out black-box tests with the goal of evaluating performance trends. Let us for now remain in the context of black-box testing.

The performance trends can be evaluated as a function of different workload mixes or workload intensities. In the latter case, we can evaluate the page response time as a function of an increasing traffic volume reaching the Web system, possibly even a little bit higher than the maximum expected workload intensity.

Figure 2.6 shows an example of performance trend evaluation in a system where the maximum expected workload intensity and the adequate performance are clearly defined. This figure considers three performance curves (P_1, P_2, P_3) and the response times obtained for the maximum expected workload intensity (MEWI in the figure). If we limit the black-box analysis to the MEWI point, we can conclude that P_1 and P_2 are acceptable, whereas P_3 does not respect the required service level. However, a trend black-box analysis evidences that even P_2 is not safe enough. Indeed, in correspondence of the maximum expected workload intensity, the P_2 curve already shows an exponential growth. In both P_2 and P_3 cases, the black-box test should be considered failed, and we should recur to the white-box testing to verify the causes of (possible) bottleneck.

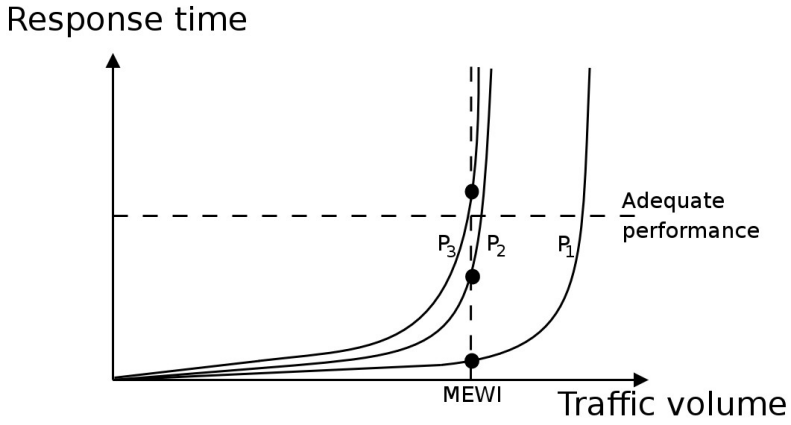

Fig. 2.6 Analysis of performance trends of the Web system

2.4.3.2 White-box testing

During the black-box testing, it is not necessary to consider the internal parts of the Web system. We can sample performance indexes outside of the Web system while it is exercised with each of the predefined workload models. On the other hand, white-box testing aims to evaluate the behavior of the internal system components of the Web platform for different client request rates (usually, around the maximum expected workload intensity). This task can always be performed after a black-box test, just to be sure that the utilization of the Web system components is well below critical levels. It becomes a necessity in two situations: when the system is violating the performance requirements (e.g., the P_3 curve in Figure 2.6), and when the trends evidenced by black-box testing raise doubts on the presence of possible bottlenecks (e.g., the P_2 curve in Figure 2.6). To find the potential or actual bottlenecks of the

Web system, it is necessary to pass to a more detailed analysis, which takes finer grained performance indexes into account. White-box testing is carried out by exercising the Web-based service with the expected workload models, and by monitoring its internals to ensure that system resource utilization does not exceed critical levels. To this purpose, we use *resource performance indexes* that measure the performance of internal resources of the Web system. They help in spotting those components of the Web system that are most utilized. Examples of resource indexes include component utilization (CPU, disk, network, central memory), but also the amount of limited software resources (such as, file and socket descriptors, process table entries). These fine-grain resource performance indexes require additional tools that must be executed during the test. Some tools are publicly available within ordinary UNIX operating systems [39, 42], but they do not provide samples for every system resource, hence some modifications to the source code (when available) is sometimes necessary.

Once the white-box testing has evidenced the nature of the bottleneck(s) affecting the Web system, it may be necessary to collect additional information to understand the causes of the problem. This allows us to plan appropriate actions for the bottleneck removal. An insufficient amount of information concerning the problem limits the range of interventions which are both effective and efficient to improve performance and reliability. To deepen the analysis, it is necessary to inspect the Web system at an even finer granularity, that of program functions. This allows us to identify the so called *hot spots* that is, critical sections of executable code consuming a significant amount of bottleneck resources.

Performance indexes at the function level are associated to the functions of each executable program including the operating system kernel. Common examples include the function call frequency and the percentage of time spent by the program into each main function. Function-level analysis requires special tools [30] that collect statistics and provide customizable views for function accesses and service times.

After the bottleneck removal step, a new testing phase follows (involving black-box and white-box testing), in order to verify that performance and reliability targets have been achieved. As outlined in Section 2.1, the whole procedure is a fix-and-test loop that may take several attempts to achieve the desired goals. In the next section, we detail the various categories of possible interventions for the removal of potential bottlenecks.

2.5 Performance improvements

Whenever a performance test fails, the Web system is not operating adequately or reliably, and proper interventions are required. As already outlined in Section 2.4, test failures can happen in different cases. First, the black-box testing can evidence a performance that is below the expected level. Second, even if the goals for "adequate performance" are met, performance can still be compromised by high utilization of some system resource that can lead to a bottleneck if the client request traffic further increases. Finally, it is often interesting to carry out capacity planning studies that test the system under *future* expected workload models. These studies tend to put more stress on the resources of the Web system, which may cause saturation and introduce

new bottlenecks that need to be removed. In this section, we discuss three main interventions for improving the performance and reliability of the Web system: *system tuning*, *scale-up* and *scale-out*.

2.5.1 System tuning

System tuning aims to improve system performance by appropriately choosing some operating system and application parameters. There are two major ways: to increase available software resources related to the operating system and some critical applications; to reduce hardware resource utilization. The typical intervention to improve the capacity of a software resource tends to raise the number of available file descriptors, sockets, process descriptors. On the other hand, sophisticated mechanisms, such as *caching* and *pooling*, are adopted to limit the utilization of critical hardware or software resources of the system. Caching avoids re-computation of information by preserving it in memory. Examples of commonly cached entities include database queries and Web pages. In resource pooling, multiple software resources are generated and grouped into a set (called *pool*) ahead of their use, so they become immediately available upon request. They are not destroyed on release, but returned to the pool. The main advantage of pooling is the reduced resource creation and destruction overhead, with a consequent saving of system resources. The TCP connections (especially, persistent TCP connections to a DBMS) are typical resources handled through a pool, because they are expensive to setup and destroy.

The size of caches and resource pools is a typical tunable parameter. Increasing the size tends to avoid re-computation (as in the case of caches) and to reduce setup/destruction overheads (as in the case of pooling). In both cases, the utilization of critical system resources is reduced. On the other hand, limitations in the available amount of memory (both main and secondary) and operating system resources (e.g., socket and file descriptors) impose a limit on the maximum size of caches and pools.

2.5.2 System scale-up

Scale-up consists in an upgrade of one or more hardware resources, but it does not adds new nodes to the underlying architecture. This intervention is necessary whenever the white-box testing evidences (the risk of) a saturated hardware resource, for example disk bandwidth or CPU power. Usually, a hardware upgrade is straightforward and does not require extensive system analysis. However, two observations are in order when performing scale-up. First, hardware upgrades are useless if an operating system resource (such as file descriptors) is exhausted. In these cases, adding hardware does not increase the capacity of the bottleneck resource. Second, performance improvements may be often obtained at lower costs through the previously discussed parameter tuning.

2.5.3 System scale-out

System scale-out interventions add nodes to the platform. This can be achieved through vertical or horizontal operations. A vertical replication deploys the logical

layers over more nodes (for example, it may pass from a two-node to a three-node architecture); an horizontal replication adds nodes to one or more layers. Both interventions improve the system performance, while horizontal replication is also at the basis for improving the reliability of the Web system. As the redesign of the platform implies non negligible costs in terms of time and money, scale-out should be pursued only when no performance improvement based on scale-up can be achieved. Furthermore, not all software technologies are well suited for scale-out. For example, from Section 2.3 we have that scripting technologies do not provide any native support for service distribution. Hence, system scale-out would imply a massive redesign of the applications supporting the Web-based service.

An even bigger scale-out intervention may be necessary when performance degradation is due to the network connecting the Web system to the Internet (the so called, *first mile*). Indeed, locally distributed Web server systems may suffer from bottlenecks at the capacity of their outbound connection [4]. Performance and scalability improvements can be achieved through a geographically distributed architecture that is managed by the content provider or by recurring to outsourcing solutions. The deployment of a geographically distributed Web system is expensive and requires uncommon skills. As a consequence, only few large organizations can afford to handle geographical scale-out by themselves. An alternative is to recur to Content Delivery Networks (CDNs) [2] that handle every issue related to Web content and service delivery, thus relieving the content provider from the design and management of a quite complex geographically distributed architecture. There are many aspects that cannot be exhaustively described in this chapter. For more details on geographically distributed Web sites the reader can refer to [40].

2.6 Case study

We present a case study that illustrates the main steps that have been introduced in Section 2.1 and detailed in the other sections. After the characterization of the Web-based service and workload models, we show a possible design and deployment of the Web system. We then carry out white-box and black-box performance testing with the goal of finding and removing system bottlenecks.

2.6.1 Service characterization and design

2.6.1.1 Workload models characterization

The considered site is an on-line shop Web site that allows users to browse a product catalog and to purchase some goods. These two main user interactions with the Web system give a picture of the type of Web resources that will be exercised in the system. In particular, the workload mix of the Web-based service is characterized by a certain amount of static Web resources, that are mainly related to product images, because most HTML documents are generated dynamically. In our case study, we assume that an external payment gateway system is used; as a consequence,

the Web system does not serve secure Web resources. The characteristics of the considered Web-based service basically correspond to a prevalent dynamic Web site (see Section 2.2.2).

2.6.1.2 Workload models characterization

The set of expected workload models for the Web site captures the most common user interactions with the Web-based service. We consider two workload models, namely *browsing* and *buying*, that have their workload mix shown in Table 2.1. The browsing workload model is characterized prevalently by product browsing actions, which exercise static and dynamic resources. The high presence of static content is motivated by the high amount of images that are shown during browsing. The buying workload model is characterized by purchase transactions which involve a high amount of dynamic resources. Table 2.1 also shows that no secure, volatile and multimedia resource are present in the workload models.

The amount of client accessing the Web site can change at different temporal scales (daily, weekly, season). However, we assume that the maximum expected workload intensity does not exceed 400 concurrent users. After this threshold the Web system may start to reject requests for connection. We will refer to this maximum workload intensity when defining the performance requirements of the Web system.

Table 2.1. Composition of the workload models

Workload model	Static resource requests	Dynamic resource requests
Browsing	60%	40%
Buying	5%	95%

2.6.1.3 Performance requirements

Once the workload models have been defined, we should set the performance requirements for each workload mix. We choose the page response time as the main system parameter. We define a first performance requirement related to the user-perceived performance. A previous study [17] shows that Web page download times exceeding 25 seconds are perceived as slow by ordinary dial-up users. However, due to the growing amount of available x-DSL and cable modem connections, we prefer to base our performance evaluation on the page response times that reflect better connected users, for example through ADSL links. J. Nielsen [36] suggests 10 seconds for each page download as the threshold for an acceptable response time in the case of high bandwidth Internet connections.

Due to the heavy-tailed distribution of the page response time, we prefer to refer to the 90-percentile of the response time. This means that the performance requirements are met if 90% of the requests have a page response time below the 10 second

threshold. For a system-oriented view, we also evaluate the system throughput in terms of served pages per second.

2.6.1.4 System design

For the design of the Web system, we must choose the software technologies for each of the three logical layers of the Web system. Due to its critical nature, we find convenient to focus on the middle layer. Since we are considering a medium-sized Web site, no extreme scalability requirements are to be met, hence we can assume that we are not interested to support this system on a highly distributed architecture. On the other hand, many pages are characterized by a fixed template with a significant amount of static HTML code. The medium size of the site and the presence of large HTML page templates suggest that a scripting technology can be a good solution for the deployment of the middle layer of the Web site.

We choose PHP [37] as the scripting language because of its efficiency and its free nature that reduces the deployment cost. PHP is easily integrated in the Apache Web server [5], which is our choice for the front-end layer. Finally, we choose MySQL [33] as the back-end. Our choice is motivated by the fact that MySQL is a free software which is widely adopted in Web sites. Furthermore, this DBMS offers adequate performance for the size of the Web site and is well supported by the PHP interpreter.

Then, we have to map the three logical layers onto the physical nodes. Scripting technologies typically lead to a two-node vertical architecture. Indeed, separation of the middle and the back-end layers on different nodes is a common choice in most medium-sized Web sites. Due to the performance requirements of the Web site, we can avoid to consider horizontally replicated architectures, which would introduce significant complexity to the middle layer software.

We can summarize the design choices for the deployment of the Web site as following. One node runs both the Apache Web server (version 2.0) and the PHP4 engine, which are used for the front-end and middle layers, respectively. The back-end layer is on a separate node running MySQL database server (version 4.0). All computers are based on the Linux operating system with kernel version 2.6.8. Each node is equipped with a 2.4GHz hyperthreaded Xeon, 1GB of main memory, 80GB ATA disks (7200 rpm, transfer rate 55MB/s) and a Fast Ethernet 100Mbps adapter.

2.6.2 Testing loop phase

We carry out an initial black-box testing to verify whether the Web system satisfies the performance requirements for all considered workload models. The test-bed architecture is rather simple, as shown in Figure 2.7: a node hosts the *client emulator*, the other two nodes compose the platform hosting the Web site.

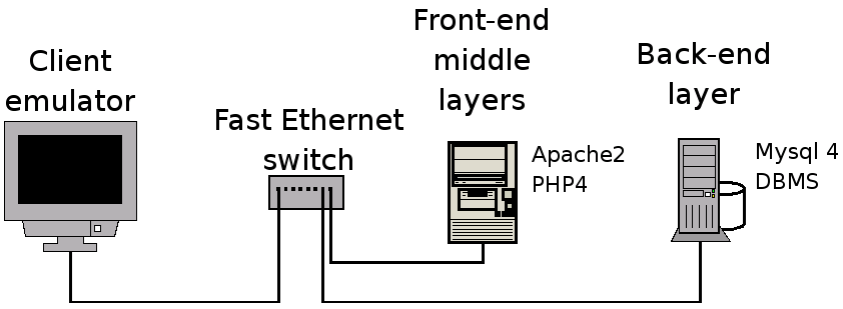

Fig. 2.7 Architecture of the test-bed for the experiments

The client emulator creates a fixed number of client processes which instantiate sessions made up 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, with an average of 7 seconds. The sequence of requests is emulated by a finite state machine that specifies the probability to pass from one Web page request to another.

To take into account the wide area network effects, we use a network emulator based on the *netem* packet scheduler [27] that creates a virtual link between the clients and the e-commerce system with the following characteristics: the packet delay is normally distributed with $\upsilon = 200ms$ and $\sigma = 10ms$, the packet drop probability is set to 1%. Bandwidth limitation in the *last mile* (that is, the client-Internet connection) is provided directly by the client emulator.

2.6.2.1 Black-box testing

Initially, we consider system level measures to determine the capacity of the Web system. We carry out tests with browsing and buying workload models and measure the system throughput and the Web page response time for different values of the client population.

Figure 2.8 shows the system throughput (measured as Web pages served per second including the embedded objects) as a function of four client populations for both browsing and buying workload models. The browsing workload model shows a nearly linear throughput increase with the user population, while the histogram of the buying workload model shows a clear throughput saturation occurring between 300 and 400 clients. Further increases of user population beyond 300 units does not improve the system throughput which remains around 40 pages per second.

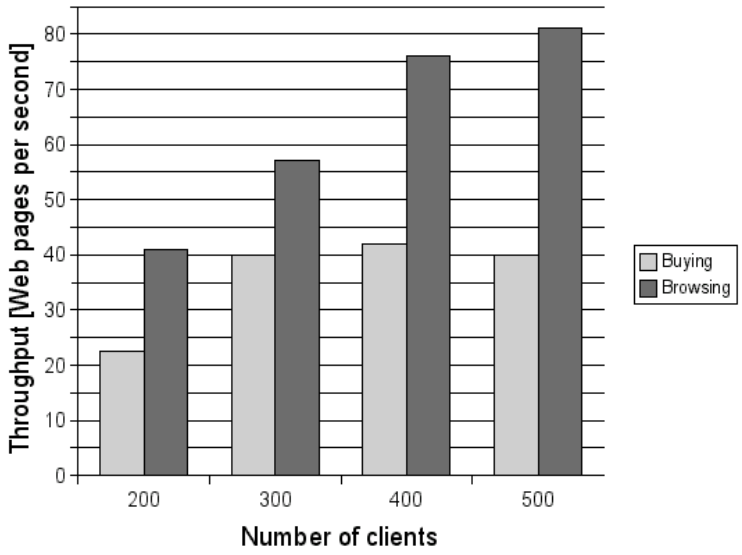

Fig. 2.8 Throughput of the Web system

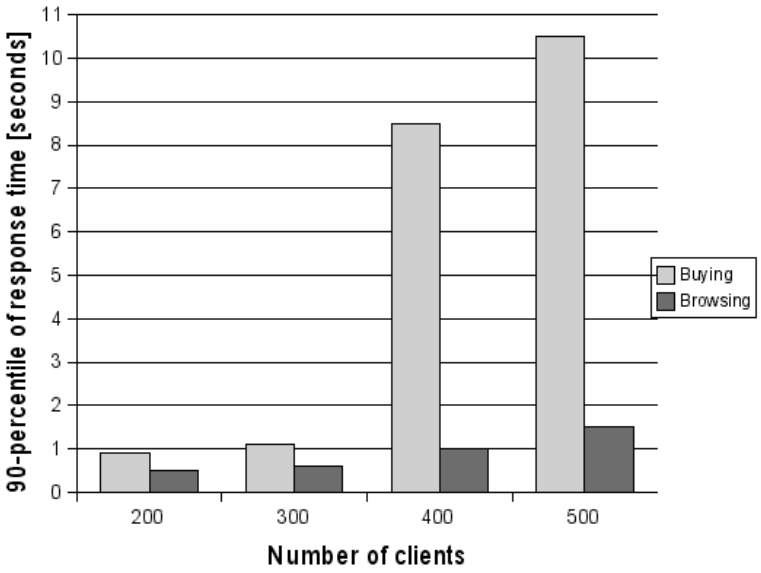

Fig 2.9 90-percentile of page response time

For different client populations we also evaluate the 90-percentile of the system response time for both workload models. The results are shown in Figure 2.9. The browsing model has s response times well below the 10 seconds threshold. On the other hand, the buying workload model shows an increase of nearly on order of

magnitude (from 0.9 to 8.8 seconds) in page response time, especially when the population passes from 300 to 400 clients. The expected performance requirement is met: a response time of 8.8 seconds is still below the 10 second threshold. However, the sudden growth in the response time, in correspondence of a critical throughput is a clear sign that between 300 and 400 clients some bottleneck occurs in the system. It is interesting also to verify whether the exponential growth trend in response time is present also for higher number of clients. For this reason we continue our black-box testing up to 500 clients. In this case we observe that the increase in response time is relatively small if compared to the growth between 300 and 400 clients. On the other hand we observe a non negligible number of errors in the service of Web pages and client request refusal. The deeper reason for this behavior cannot be explained by black-box testing but require a further analysis. The negative trend evidenced by the black-box testing and the performance value close to the threshold of adequate performance, suggest to plan and undertake some countermeasure to improve the system performance, as described in Section 2.4. To understand the directions that must be followed, it is necessary to carry out a white-box testing.

2.6.2.2 White-box testing

We now present the results of a white-box testing, in which we investigate the utilization of the Web system internal resources. The main goal is to investigate the causes of performance degradation that is evidenced by the black-box testing, when the system is subjected to the buying workload model for a client population between 300 and 400 units. Furthermore white-box testing allows to understand the causes of the errors in client request service observed for higher client population (e.g., 500 users). We chose to start the white-box analysis for a client population of 400 clients, because it is around or just after the knee of the performance curve of the Web system.

The finer grained performance evaluation takes into account resource performance indexes such as CPU, disk, and network utilization. Table 2.2 shows the results of the white-box testing for different resources. Utilization values are reported as the sample averages throughout the entire test duration. From this table, it is easy to detect that the system bottleneck is represented by the CPU of the node hosting the back-end layer.

This is confirmed by the curve in Figure 2.10, showing the CPU utilization of the back-end node during the experiments (the horizontal dotted line represents the mean value). A CPU utilization at 0.9 is a clear sign of resource over-utilization that may be at the basis of a system bottleneck. The 80-20% ratio between the times spent in the user and kernel mode suggests that the application level computations on the DBMS are much more intensive than the overhead imposed by the management of the operating system calls.

Table 2.2. Resource utilization (white-box testing)

Performance index	Front-end and Middle-layer	Back-end layer
• CPU utilization	0.31	0.90
- user mode	0.21	0.76
- kernel mode	0.10	0.14
• Disk utilization	0.003	0.015
• Network interface utilization	0.012	0.002

White-box tests show also an initially unexpected result: even if the bottleneck is related to the DBMS, the disk utilization is quite low (0.015). The motivation for this result must be found in the size of the database which in large part fits in the 1Gbyte main memory. We conclude that for this case study the disk activities do not represent a potential bottleneck of the back-end node hosting the database server. Instead, the bottleneck cause must be found in the CPU of the back-end node. Our experiments confirm an interesting trend: with the hardware improvements even at the entry level, it is becoming common for medium-size e-commerce databases to fit for a large part in main memory.

White-box testing allows also to provide an explanation for the errors occurring when the system is subject to heavier load, as in the case of a population of 500 clients. In this case the finite queue of pending connection requests gets saturated. As a consequence further client connection attempts get refused and the actual amount of requests served by the system is only slightly higher than in the case of 400 clients. The final consequence is that the Web system tends asymptotically to saturation, as shown by the black-box analysis. However, since the saturation of the queue of pending connections occurs after the bottleneck on the back-end node, in the following we will focus on this latter issue, which has a more significant impact on performance.

The next step aims to remove the cause of the system bottleneck. To this purpose, it is necessary to identify the causes of the problem through an even finer analysis that is based on operating system monitors. These tools allows us to identify the software component that is utilizing the large part of the CPU in user mode. As there is only one major application running on the back-end layer node, we can deduce that the MySQL server process is the source of the bottleneck. However, if we limit the analysis at the process granularity level, we do not have any hint. It is necessary to evaluate finer grained indexes, at the function level. These indexes will permit to identify the source of the problem and fully explain the causes of the inadequate performance.

We present the results of the same experiment, that is executed under an efficient operating system profiler [30] on the node hosting the DBMS. The profiler output shows more than 800 DBMS functions, hence a detailed analysis of all function access times and frequencies is quite difficult, and even useless. The idea is to focus

on the functions that use more CPU time, while we aggregate the other functions that are not significant for the bottleneck analysis. The evaluation shows that most CPU time is consumed by consistency checks on store data, hence we can conclude that the real cause of the bottleneck is represented by the asynchronous I/O subsystem adopted in the MySQL process.

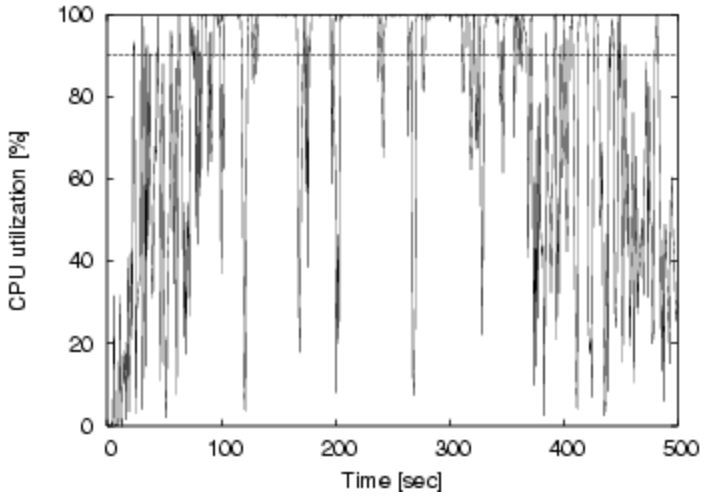

Fig. 2.10 CPU utilization of the back-end layer node

2.6.3 System consolidation and performance improvement

From the results of the white-box test we have that the asynchronous I/O operations on the DBMS require more CPU power than what is currently available. We have three possible interventions to address this issue: system scale-out, system scale-up and system tuning. The goal now is to understand the most appropriate solution.

Scaling-out the system is not the best approach to solve the problem. Vertical replication is not effective in reducing the load on the back-end node because it does not allow to distribute the DBMS over multiple nodes. The only viable solution would be an horizontal replication of the DBMS, but MySQL has no native support to manage consistency in a database distributed over multiple nodes. Furthermore, a similar intervention would require a mechanism to distribute queries over the multiple back-end nodes. This means a complete redesign of the back-end layer and, consequently, of significant portions of the middle-layer. For these reasons, we avoid interventions based on system scale-out.

Scale-up is a viable solution: upgrading the existing hardware is a straightforward approach, in particular if we increase the CPU speed of the back-end node. However, we find interesting also to investigate the hypothesis of tuning the parameters of the DBMS system. As the problem is on the asynchronous I/O subsystem, we could try to reduce the asynchronous I/O activity by decreasing the number of buffer accesses. For example, this can be accomplished by increasing the size of the query cache.

After this intervention, we re-evaluate the system performance with a second test phase. We find that the CPU utilization on the back-end node passes from 0.9 to 0.6. As expected, reducing the CPU bottleneck on the back-end node improves the performance of the overall system. Figure 2.11 reports the cumulative distributions of the response time of the system before and after the reconfiguration of the system parameters. This figure confirms the validity of the intervention because the 90 percentile of the response time drops from 8.8 to 3.1 seconds after the database tuning.

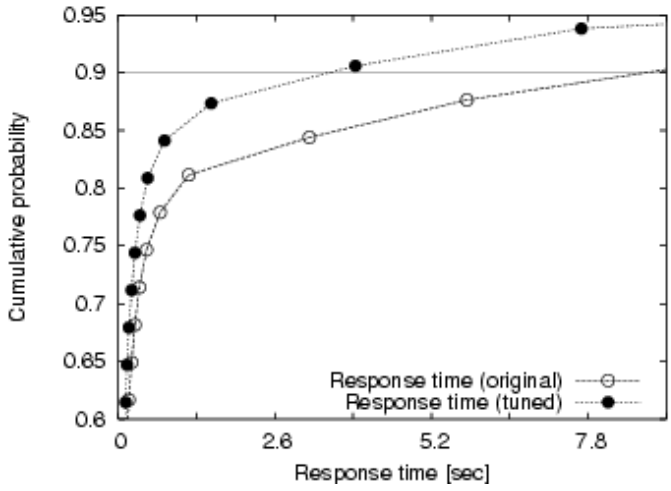

Fig. 2.11 Cumulative distribution functions of the system response time

2.7 Conclusions

Web sites are becoming a critical component of the information society. Modern Web-based services handle a wide, heterogeneous galaxy of information including news, personal information, multimedia data. Due to their growing popularity and interactive nature, Web sites are vulnerable to the increase in the volume of client requests, which can hinder both performance and reliability. Thus, it is fundamental to design Web systems that can guarantee adequate levels of service at least within the expected conditions of traffic.

Throughout this chapter we presented a methodology to design mission-critical Web systems that take into account performance and reliability requirements. The proposed approach is conceptually valid for every Web site, although we focus mainly on systems that are characterized by prevalent dynamic content because this widely diffused category presents interesting design and implementation challenges.

Besides the design process, we describe the issues related to performance testing by showing the main steps and the goals of black-box and white-box performance tests. We finally consider some interventions that can be carried out whenever performance tests are not satisfactory. After the identification of the causes of violation, we present the main interventions to improve performance: system tuning, system scale-up and system scale-out. The book chapter is concluded with a case study in which we apply the proposed methodology to an medium-size e-commerce site.

Acknowledgement

The authors acknowledge the support of MIUR in the framework of the FIRB project "Performance evaluation of complex systems: techniques, methodologies and tools" (PERF).

References

1.Active Server Pages (2004) http://msdn.microsoft.com/asp
2.Akamai Technologies (2005) http://www.akamai.com.
3.Andreolini M., Colajanni M., Morselli R. (2002) Performance study of dispatching algorithms in multi-tier web architectures. In: ACM Sigmetrics Performance Evaluation Review, 30(2):10-20.
4.Andreolini M., Colajanni M., Nuccio M. (2003) Kernel-based Web switches providing content-aware routing. In: Proceedings of the 2nd IEEE International Symposium on Network Computing and Applications (NCA), Cambridge, MA.
5.Apache Web server (2005) http://httpd.apache.org.
6.Arlitt M. F., Jin. T. (2000) A workload characterization study of the 1998 World Cup Web site. IEEE Network, 14(3):30–37
7.Arlitt M. F., Krishnamurthy D., Rolia J. (2001) Characterizing the scalability of a large scale Web-based shopping system. ACM Trans. on Internet Technology 1(1):44–69
8.Arlitt M. F., Williamson C. L. (1997) Internet Web servers: Workload characterization and performance implications. IEEE/ACM Trans. on Networking, 5(5):631–645
9.Barford P., Crovella M. (1998) An architecture for a WWW workload generator. In: Proceedings of SIGMETRICS, Madison, WI.
10.Barford P., Crovella M. (1998) Generating representative Web workloads for network and server performance evaluation. In Proc. of Sigmetrics 1998, pages 151–160, Madison, WI
11.Cardellini V., Casalicchio E., Colajanni M., Yu P.S. (2002) The state of the art in locally distributed Web-server systems. In: ACM Computing Surveys, 34(2):263-311.
12.Cardellini V., Colajanni M. Yu P.S., (1999) Dynamic load balancing on Web server systems, IEEE Internet Computing
13.Cardellini V., Colajanni M., Yu P.S. (2003) Request redirection algorithms for distributed Web systems. In: IEEE Transactions on Parallel and Distributed Systems, 14(4):355–368.
14.Cecchet E., Chanda A., Elnikety S., Marguerite J., Zwaenepoel W. (2003) Performance comparison of middleware architectures for generating dynamic Web content. In: Proceedings of the ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil.

15.Chen H., Mohapatra P. (2002) Session-based overload control in QoS-aware Web servers. In: Proceedings of IEEE Infocom, New York, NY.
16.Chiu W. (2000) Design pages for performance. IBM High Volume Web Site white papers.
17.Chiu W. (2001) Design for scalability: an update. IBM High Volume Web Site white papers.
18.Coarfa C., Druschel P., Wallach D. (2002) Performance analysis of TLS Web servers. In: Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA.
19.The Apache Cocoon Project (2005) http://cocoon.apache.org.
20.Cold Fusion (2004) http://www.coldfusion.com
21.Darwin Streaming Server, http://developer.apple.com/darwin/projects/streaming/
22.Edge Side Includes, ESI (2004) http://www.esi.org
23.Elnikety S., Nahum E., Tracey J., Zwaenepoel W. (2004) A method for transparent admission control and request scheduling in e-commerce Web sites. In: Proceedings of the 13th International Conference on World Wide Web, New York, NY.
24.Fraternali P. (1999) Tools and approaches for developing data-intensive Web applications: a survey. In: ACM Computing Surveys, 31(3):227–263.
25.Goldberg A., Buff R., Schmitt A. (1998) Secure Web server performance dramatically improved by caching SSL session keys. In: Proceedings of SIGMETRICS, Madison, WI.
26.Gray J., Helland P., O'Neil P. E., Shasha D. (1996) The dangers of replication and a solution. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada.
27.Hemminger S. (2004) Netem home page: http://developer.osdl.org/shemminger/netem.
28.Apple iTunes (2005) http://www.apple.com/itunes.
29.Java 2 Platform Enterprise Edition, J2EE (2004) http://java.sun.com/j2ee
30.Levon J. (2004) Oprofile: a system profiler for Linux. http://oprofile.sourceforge.net
31.Menascè D. A., Almeida V. A. F., Riedi R., Pelegrinelli F., Fonseca, R, Meira V. (2000). In search of invariants for e-business workloads. In Proc. of 2nd ACM Conf. on Electronic Commerce, Minneapolis, MN
32.Menascé D. A., Barbarà D., Dodge R. (2001) Preserving QoS of e-commerce sites through self-tuning: a performance model approach. In: Proceedings of the 3rd ACM conference on Electronic Commerce, Tampa, FL.
33.MySQL database server (2005) http://www.mysql.com.
34.Nahum E., Rosu M. C., Seshan S., Almeida J. (2001) The effects of wide-area conditions on WWW server performance. In: Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Cambridge, MA.
35.Netcraft (2005) http://www.netcraft.com/survey/archive.html.
36.Nielsen J. (1994) Usability Engineering. Morgan Kaufmann Publishers Inc., San Francisco, CA.
37.PHP scripting language (2005) http://www.php.net.
38.PostgreSQL database Server (2005) http://www.postgresql.org.
39.Procps: the /proc file system utilities (2005) http://procps.sourceforge.net.
40.Rabinovic M., Spatscheck O. (2002) Web caching and replication. Addison Wesley Publishing.
41.Rabinovich M., Xiao Z., Douglis F., Kalmanek C. (2003), Moving edge side includes to the Real Edge – the clients, Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems
42.Sar: the system activity report (2005) http://perso.wanadoo.fr/sebastien.godard.
43.Sculzrinne H., Fokus G.M.D., Casner S., Frederick R., Van Jacobson (1996) RTP: A transport protocol for real-time applications, RFC 1889
44.The Tomcat servlet engine (2005) http://jakarta.apache.org/tomcat.
45.Vallamsetty U., Kant K., Mohapatra. P. (2003) Characterization of e-commerce traffic. Electronic Commerce Research, 3(1-2):167–192.

Authors biography

Mauro Andreolini

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", January, 2001. 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 locally 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 and Reggio Emilia. 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

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, in addition to several national conferences. 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 AAA-IDEA Workshop. 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

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