Improving the availability of web services *

D. Cotroneo[1,2], M. Gargiulo[2], S. Russo[1,2], G. Ventre[1,2]
[1]Universitá degli Studi di Napoli "Federico II"
Dipartimento di Informatica e Sistemistica
Via Claudio 21, 80125 - Napoli, ITALY
[2]Consorzio C.I.N.I.
Laboratorio ITEM
Via Diocleziano 328, 80124 - Napoli, ITALY
{cotroneo, sterusso, giorgio}@unina.it
mauro.gargiulo@napoli.consorzio-cini.it

ABSTRACT

In order to maintain the popularity and reputation of a web site, the quality of service perceived by users, especially the service availability, is a success factor. A service that is frequently unavailable may have negative effects on the reputation of the service provider, or result in loss of business opportunities. From the user's perspective, a service that exhibits poor quality is virtually equivalent to an unavailable service. In this work, we present the overall architecture and the evaluation of a middleware infrastructure which provides quality-of-service differentiation among classes of communication-bound processes. By communication-bound processes we mean processes whose activity is typically dominated by network communication, e.g. a video server. The proposed architecture supports different classes of service, each with different quality attributes concerning the network data delivery performance. In particular, the architecture is able to provide a class of service, namely *guaranteed service class*, which is suitable for increasing the service availability for a group of premium users, especially in overloaded servers (in absence of external faults).

Keywords

Class-of-Service, Web-service, Availability, Real-time Operating Systems

1. INTRODUCTION

The Internet world is moving toward a scenario where users and applications have very diverse service expectations, making the current best-effort model inadequate and limiting. In fact, new web applications demand for delivery of multimedia data in real-time (e.g. streaming stored video and audio), and the information transfer via the Internet is becoming one of the principal paradigm for business: electronic sales, banking, finance, collaborative work, are examples of this. In this scenario, in order to maintain the popularity and reputation of a web site, the quality of service perceived by users, especially the service availability, is a success factor [5]. The principal QoS attributes that users perceive include those related to the service availability and timeliness: a service that is frequently unavailable may have negative effects on the reputation of the service provider, resulting in loss of business opportunities. From the user's perspective, a service that exhibits poor quality is virtually equivalent to an unavailable service.

The performance perceived by the users of a Web service depends on the network infrastructure (possibly QoS-enabled) but especially on the management of the servers' resources [13]. It is thus desirable that network servers (e.g., Web, Video on Demand, and FTP servers) should be able to differentiate their services in a variety of classes, replacing the current simple best-effort paradigm. This leads to a model in which applications and users are treated differently, in a way that best meets their quality and pricing constrains. This paper presents the overall architecture and the experimental evaluation of an operating system extension for service differentiation of communication-bound processes. The architecture provides server application developers with a communication library (similar to the standard socket), named cosSocket (*class of service-enabled Socket*), which is able to realize different classes of service using features of a real-time operating system. A service differentiation scheme can be applied to different applications, or to different users in the context of the same application. Service differentiation is obtained by assigning different CPU time-slices to applications I/O tasks. The underlying idea is that it is possible to decrease time-slice assigned to a given process in order to reduce its communication throughput, freeing server's resources in favor of processes with a better class of service. Real-time features are achieved through a Rate Monotonic Algorithm for CPU scheduling, included

*This work has been performed under the financial support of Italian Ministry of Education, University and Research (MIUR) under grant "LABNET2". The work of D. Cotroneo and S. Russo has also been supported by MIUR within the project "MUSIQUE: Infrastructure for QoS in Web Multimedia Services with Heterogeneous Access".

.

in the TimeSys Linux/RT kernel [*www.timesys.com*], which also offers a complete support for POSIX threads.

The paper is organized as follows. In section 2 we introduce the web service correctness and related problems. The failure mode assumptions we adopted is presented in section 3. In section 4 we describe the overall architecture; experimental results are provided in section 5. Section 6 discusses related work in this field. Finally, section 7 provides some concluding remarks related to the obtained results, along with information on future work.

2. WEB SERVICE CORRECTNESS

Before illustrating the design and the implementation of the proposed middleware infrastructure, it is worth clarifying the definition of the web service availability, starting from the position stated in [14] by D. Powell. Availability deals with the readiness for correct service. In particular, it is a function $A(t)$, which is the probability that the system is operational (i.e., delivers the correct service) at instant of time t. This function quantifies the alternation between deliveries of correct service and incorrect service. A system can fail to deliver a correct service due to the following reasons:

- the presence of faults, caused by system errors;
- the presence of overloading condition, i.e. the server is so much busy that it is not able to deliver a correct service.

Throughout this paper we focus only on system failures, stemming from overload conditions. This kind of failures is strictly tied with the delivered quality of service because, if the QoS falls down under a certain threshold, the service can be considered unavailable. The architecture we propose provides an efficient and flexible resources management strategy, which aims at improving the quality of the delivered service, reducing the QoS degradation perceived by some premium users. The result is a higher probability of delivering to these users a correct service, improving the system availability. The architecture does not prevent system from hardware/software faults, hence it does not guarantee the service availability. In order to achieve this further goal, a redundant scheme has to be also implemented, as described in section 7.

As stated in [14] and in [4], in order to analyze the availability of a system it is essential to clarify what does correct service mean. Starting from definition given by Powell, the service delivered by a system can be defined in terms of a sequence of service items $s_i, i = 1, 2....$, each characterized by a tuple $< vs_i, ts_i >$, where vs_i is the value or content of service item s_i and ts_i is the time or instant of observation of service item s_i. Assuming the presence of an omniscient observer that has a complete knowledge of the specified sequence of service items the system should deliver, a service s_i is defined correct if:

$$(vs_i \in SV_i)\, and\, (ts_i \in ST_i)$$

where SV_i and ST_i are respectively the specified sets of values and times for service item s_i. For a general system, SV_i and ST_i are functions of the sequence of system inputs. As far as modern web-based systems with QoS constraints are concerned, this definition is certainly suitable, but sets SV_i and ST_i have to be extended.

Indeed, modern web-based systems are implemented over a QoS-enabled network. In this context, the term QoS is related to the quality of communication service, such as a certain value of packet loss, latency, jitter, and assured bandwidth, appearing at the communication endpoints like a point-to-point connection or a virtual "leased line" with the requested quality attributes. In this scenario, it should be possible to provide the same service with different quality attributes to several classes of users. In this way, the correctness of a service also depends on the specified class of users which request it. We can thus define the correctness of a web service, for a certain class j of users, as:

$$(vs_i \in SV^*_{i,j})\, and\, (ts_i \in ST^*_{i,j})$$

where:

$$SV^*_{i,j} = f(SV_i, CU_j)$$

$$ST^*_{i,j} = f(ST_i, CU_j)$$

and where CU_j represents the class j to which the user belongs. It is thus desirable to have an architecture in which applications and users are treated differently, in a way that best meets their quality and pricing constrains. The proposed middleware architecture, using features of a real-time operating system, is able to realize different classes of service inside the web server too. A service differentiation scheme is provided to different applications, or to different users in the context of the same application.

3. FAILURE MODE ASSUMPTIONS

This section gives a formal definition of the failure modes of a web server we adopted. By web server it is meant a server, such as multimedia or HTTP server, that provides its services via a web infrastructure. According to [14], a failure mode is defined in terms of an assertion on the sequence of value-time tuples that a server is supposed to deliver. Let us assume the following class of users exist: C_n(normal user), C_m (medium user), C_p (premium users). Assertions may be defined in the value domain and in the time domain. Effects of value errors are not considered afterward. As already mentioned, we investigate failures caused by server overload conditions, i.e. the server process or host is too busy for delivering the correct service to a certain class of user.

3.1 Timing errors assertion

These assertions are the most important in the considered context.

- No timing errors can occur:
  $\tau_{none} := \forall i, \forall j\ ts_i \in ST_{i,j}$
- Omission errors can occur:
  $\tau_O := \forall i, \forall j\ (ts_i \in ST_{i,j})\, or\, (ts_i = \infty)$
- Late timing errors can occur:
  $\tau_L := \forall i\ (\forall j,\quad ts_i \in ST_{i,j})\, or\, [\exists j \in \{m, p\} : (ts_i > maxTime_j)]$

The omission error assertion depicts a fail silent behaviour: in such a situation if the system does not supply the service s_i in ST_{ij} it will never supply it at all. Similary, a late timing error can occur if a service item s_i is delivered at a medium

or premium user after a threshold, named $maxTime$, depending on user requirements.

It should be emphasized that late timing errors can occur only when medium (C_m) and premium users (C_p) request the service. This means that a service delivered to users belonging to class C_n, is correct even though it is delayed for a time greater than $maxTime$.

4. OVERALL SYSTEM ARCHITECTURE

As mentioned, we focus on Internet-based data delivery servers (Web, FTP, Video on Demand servers). For these kinds of servers, controlling I/O activities is essential to achieve a pre-determined behaviour. We propose an architecture which provides differentiated communication services according to a number of service classes. The real-time scheduler assigns to each service class a CPU amount depending on its service level. By doing so, it is possible to schedule processes in a deterministic way. However, assigning a service level to the entire process does not ensure real-time communication. In fact, the performance of a communication-bound process mainly depends on the scheduling of its I/O tasks, as indicated in [7, 8]. The architecture we propose is in charge of managing I/O activities of all processes residing on the end-system. In fact, process I/O tasks consist of a sequence of system call invocations which require the execution of an operating system thread serving the request. Our strategy relies on the capability of controlling the number of system calls issued for requesting I/O tasks.

The proposed architecture is able to completely separate I/O from the CPU activities, by providing application developers with a new communication library (cosSocket) similar to the standard TCP/IP socket library. Once separated these activities, I/O tasks can be scheduled by the real-time kernel. As for the design methodology, we adopted an object-oriented approach, namely *Concurrent Object Modeling and Architectural Design Method* (COMET), particularly suited for designing concurrent and real-time distributed systems [9]. The static model for the overall architecture is depicted in figure 1.

As shown in figure 1, user applications can create one or more instances of class cosSocket. From user point of view, such a class is able to perform I/O operations with a specified quality of service. The cosManager, which inherits from a POSIX thread class, is in charge of handling an instance of cosSocket. In other words, by means of cosSocket each application delegates, or defers, all the I/O activities to an instance of cosManager. I/O operations of cosManager are then performed using standard socket libraries. The activities of cosManager, which are mainly I/O calls, are optimized and then scheduled by the real-time kernel by means of the cosDaemon. This daemon is the only architecture's entity, capable of using real-time features in order to control all cosManagers present on the end-system. The cosDaemon is also in charge of implement admission control policies for guaranteed services.

We defined a class service model which consists of two kinds of service classes: Adaptive and Guaranteed. They are presented in the following subsections.

4.1 Adaptive class service

By adaptive we mean a service class that can be requested without any admission control mechanism [2]. According to this class definition, we allocate CPU shares in a weighted way. This means that we set preliminary n weights, $W_1 < W_2 < ... < W_n$, associated to each of n classes (class n has the highest priority). The adaptive service does not provide hard guarantee on the effective throughput of each process; however, it allows to define several classes, providing a guaranteed differentiation between them on the basis of assigned weight.

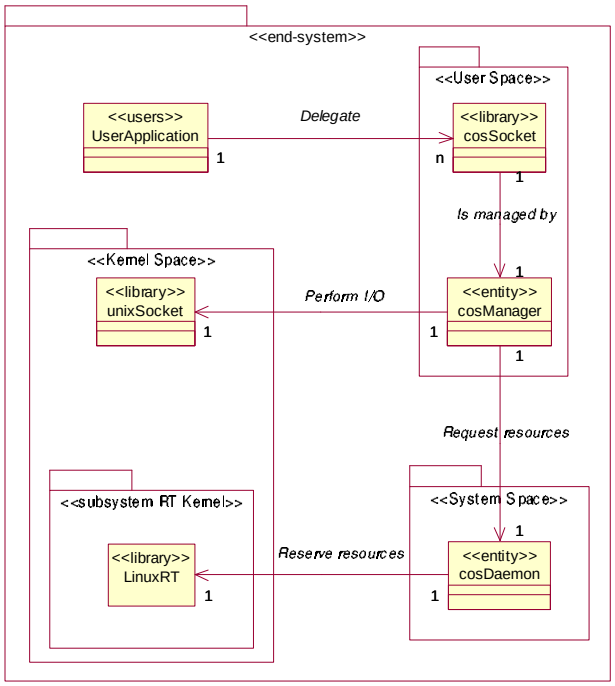

Figure 1: COMET Static Diagram of the proposed architecture.

4.2 Guaranteed service class

By guaranteed service class we mean a class subject to an admission control policy. In this case, each instance requires a specific throughput. The request can be accepted or rejected according to the specific policies implemented in the admission control module. If accepted, the service has to be guaranteed by the system during all its life cycle. This kind of service is particularly suitable for applications that require a constant throughput (e.g. multimedia applications) or for satisfying a group of premium users, leaving the service always available to them independently of the system workload (in absence of external faults).

The mechanism for providing Guaranteed services class is implemented by a self-regulating utilization control loop. The throughput control loop determines the CPU amount necessary for obtaining a Constant Bit Rate (CBR). Let y_0 the desired throughput, and y the current throughput obtained by the cosManager. For the sake of simplicity, the cosManager was modeled like a "black box". The value e is the "throughput error", $e = y_0 - y$. The cosDaemon, which acts as the controller, samples the current throughput y, and computes the corresponding error e at fixed time intervals, then produces an output, u, that regulates the CPU

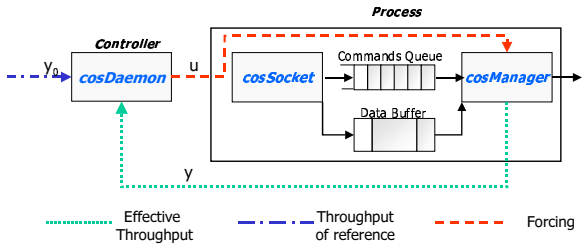

Figure 2: Self-regulating control loop for providing Guaranteed service.

to be assigned to the cosManager. We used a proportional-integral (PI) controller in our loop. The controller produces an output that is proportional to the last error and to the sum of the previous m errors. At each sampling time the controller performs the following computation, diminishing the medium quadratic error: $u = u + k * e$ where k is a constant. The adopted scheme is illustrated in figure 2.

4.3 Implementation models

As far as implementation is concerned, two main issues have been addressed: the synchronization mechanism between cosSocket and cosManager, and data buffer management. The solution of these problems resulted in three different implementation models, as shown in [6]:

- *Synchronous model*;
- *Asynchronous model*;
- *Asynchronous Aggregated model*.

We implemented all the three models and evaluated them in order to investigate which of ones is best suited for the considered applications. Experimental results, reported in [6], show how best performance are obtained with the last one. Therefore we adopted the Asynchronous Aggregated model, and results discussed in section 5 refer to this model.

5. AVAILABILITY EXPERIMENTS

In this section we present measurements which aim to demonstrate how the proposed middleware architecture is able to improve availability of service delivered to the premium user class.

The testbed used is composed of three different kinds of COTS PC, as described in the following:

- One HTTP server (named Tigri): is a Pentium III at 600 Mhz with 256 Mb of RAM and Linux/RT by Timesys installed.
- One HTTP client (named Eufrate): is a Pentium III at 600 Mhz with 256 Mb of RAM and Linux/RT by Timesys installed.
- Five HTTP clients: Celeron 700 Mhz with 128 Mb of RAM and Windows 2000 Professional installed.

All PCs are on the same LAN at 100 Mbit/s full switched.

On the Server Tigri we installed our architecture. We also used a simple HTTP-server application which replies to the client requests on different port numbers, one for each class

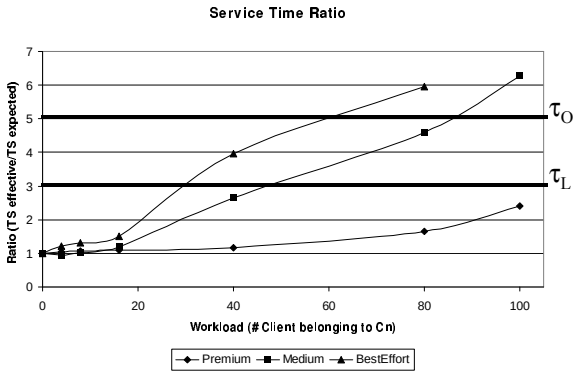

Figure 3: Performance measurements in presence of variable workload.

of service, as previously described: C_n(normal user), C_m (medium user), C_p (premium users).

The Windows clients are used only to generate the workload on the target server. To this aim, each Client run one or more instances of a http-client application requiring a file transfer service belonging to the C_n class. The same HTTP-client was also used on the Linux client machine, where measures are taken, in order to require a new service belonging each time to one of the three possible classes. We used a Linux client with the real-time extension in order to obtain a high-precision timer. All requests issued concern the same service: the transfer of a file of 10 Mbyte. The Windows clients repeat such a request in a cyclic way, to obtain a constant workload for all tests. Each test was repeated five times and the final result is the average value.

Tests were performed by evaluating the bandwidth obtained by the Linux client and the correspondent service time, with different workload due to the variable number of C_n connections. The C_n and the C_m classes are handled in a adaptive way, so in these cases the architecture do not provide a guaranteed quality of the service delivered, but only a service differentiation. Effective performance, for users belonging to C_n and C_m classes, depend on the total workload. Instead, at each premium user (C_p) is assigned an instance of the Guaranteed service class with a throughput of 5 Mbit/s.

We were interested to effective service time and bandwidth, establishing for both an expected value in absence of workload. Then we made test and compared real value with those expected. The results, for each service class, are depicted in figure 3 as the ratio between effective and expected service time.

As the figure shows, in presence of operating system trashing conditions (i.e., increasing the number of connection requests) there are differences between expected and effective values (ratio greater than 1). This difference increases more rapidly for users belonging to C_n and C_m classes than for users belonging to C_p one.

In particular, we assumed to have a Late Timing Error, τ_L, when the effective service time is three time greater then the expected one, and to have an Omission Error, τ_O, when this ratio is greater than five. Althought, as explained in section 3, these tresholds are defined only for medium and premium users, they can be useful to compare the service

degradation between the different classes.

According to these rules, the C_m service experiences a Late Timing Error with a workload of about 48 C_n requests instead of only 25 needed for the C_n service. Similarly, an Omission Error occurs with 87 requests for C_m and only 59 for C_n.

On the contrary, a workload of 100 requests is not enough to cause a LateTiming Error for the Premium user class C_p.

It is clear from the figure that the proposed architecture is able to prevent the class C_p from overload conditions by guaranteeing the availability of the service even in the case of server overload.

6. RELATED WORK

Quality of Service provisioning for data delivery and real-time applications have received considerable attentions in [1] and [2]. There are been appreciable progresses in QoS support separately for Web Server [1]. Many works like [13] as well as our previous experience in quality of service support [7], highlights needs to service differentiation even in the end system. Different architectures have been proposed and implemented in order to support QoS guarantees in the end-system. For example, in [12] are proposed some architectural mechanisms to manage communication resources for guaranteed-QoS connections, and in [10, 15] has been addressed the problem of scheduling real-time applications on general-purpose Operating System in order to provide different classes of communication services. Both architectures did not address the implementation issues of a mechanism to control the bandwidth assigned to different class of service. In all revised works, resources control was used to increase performance or to provide class differentiation, without considering the lack of availability due to a poor control of communication QoS.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we focused on Internet-based data delivery services (e.g., services provided by Web, FTP, and video-on-demand servers). These services are run by processes whose activity is typically dominated by network communication; we called them communication-bound processes. We presented an overall description of an operating system extension for quality-of-service differentiation among classes of communication-bound processes. Our strategy relies on the capability of controlling the I/O activity performed by applications. We defined three Class of Service corresponding to different priorities in I/O-resources utilization. An extended evaluation demonstrated that such an architecture is useful to improve the satisfaction of premium users, increasing the effective availabilty in presence of system overload. Finally, we are currently investigating the behaviour of the cosSocket architecture with respect to other performance parameters as response time and jitter. We are also evaluating the influence of architecture setup on this parameters.

8. REFERENCES

[1] T.F. Abdelzaher, N. Batti, "Web Server QoS Management by Adaptive Control Delivery", in *International Workshop on Quality of Service (IWQOS'99)*, London, UK, June 1999.

[2] T.F. Abdelzaher, K.G. Shin, "End-host architecture for qos-adaptive communication", in *Proc. of IEEE Real Time Technology and Applications Symposium*, Denver, Colorado, June 1998.

[3] C. Aurrecoechea, A. Campbell, L. Hauw, "A survey of QoS architecture", in *4th IFIP International Conference on Quality of service*, Paris, France, March 1996.

[4] A. Bondavalli, L. Simoncini, "Failure Classification with respect to Detection", in *Esprit Project N.3092 (PDCS), 1st Year Report*, IEEE-CS, Los Alamitos, CA (USA), May 1990.

[5] S. Chandra, C. Ellis, A. Vahdat, "Differentiated Multimedia Web Services Using Quality Aware Transcoding", in *Proc. 19th Annual Joint Conference Of the IEEE Computer And Communications Societies (INFOCOM'00)*,December 2000.

[6] D. Cotroneo, M. Ficco, M. Gargiulo, S. Russo, G. Ventre, "Service Differentiation of Communication-bound Processes in a real-time Operating System", in *Proc. of 7th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2002)*, San Diego, CA, Jenuary 2002.

[7] D. Cotroneo, M. Ficco, S. Romano, G. Ventre, "Bringing Service Differentiation to the End System", in *Proc. of the IEEE International Conference on Networks (ICON'00)*, Singapore, Oct. 2000.

[8] D. Cotroneo, M. Ficco, G. Ventre, "Bringing Service Differentiation to the End System", in *Proc. of the 8th International Workshop on Interactive Distributed Multimedia Systems*, Lancaster, September 2001.

[9] Hassan Gomaa, "Design Concurrent, Distributed, and Real-Time Applications with UML ", Addison Wesley, Object Technology Series, 2000.

[10] D. Ingram, "Soft Real-Time Scheduling for general Purpose Client-Server Systems", in *Proc. of the IEEE Workshop in Operating System*, Aug. 1999.

[11] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, J. Hansen, "A scalable solution to Multi-Resource QoS problem", in *the 20th IEEE Real-Time Systems Symposium*, Aug. 2000.

[12] A. Mehra, Kang G. Shin,"Structuring Communication Software for Quality-of-Service", in *IEEE Transactions on Software Engineering,* (Vol. 23, No. 10), pp. 616-634, October 1997.

[13] K. Nahrstedt, J. Smith, "The QoS Broker", in *Proc. of the IEEE Multimedia Spring* 1995,Vol.2, No.1, pp. 53-67.

[14] D. Powell, "Failure Mode Assumptions and Assumption Coverage", in *Proc. of the 22nd International Symposium on Fault-Tolerant Computing (FTCS-22)*, IEEE-CS, Los Alamitos, CA (USA), 1992.

[15] I. Stoica, "A Proportional Share Resource Allocation Algorithm For Real-Time, Time-Shared Systems", in *Proc. Of IEEE Real-Time Systems Symposium*, December 1996.