Service Differentiation of Communication-bound Processes in a real-time Operating System

D. Cotroneo, S. Russo, G. Ventre
Universitá degli Studi di Napoli "Federico II"
Dipartimento di Informatica e Sistemistica
{cotroneo, sterusso, giorgio.ventre}@unina.it

M. Ficco, M. Gargiulo
Consorzio Interuniversitario Nazionale per l'Informatica
ITEM - Laboratorio Nazionale di Informatica e Telematica Multimediali
{massimo.ficco, mauro.gargiulo}@napoli.consorzio-cini.it

Abstract

The majority of today's Internet-based services are generally not concerned about the level of Quality of Service (QoS) presented to their users. For many such services, however, the QoS perceived by users is becoming a critical success factor. The main QoS attributes include those related to the service availability and timeliness. Ensuring them is essential to many services. In our opinion, this has to be achieved not only by providing services with appropriate access bandwidth, or through QoS awareness of the network communication protocols used, but also by means of a differentiation of the usage of system resources by server processes. In this paper, we focus on Internet-based multimedia 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 call them communication-bound processes. We present the design and implementation of an operating system extension for quality-of-service differentiation among classes of communication-bound processes. The system allows to define classes of services with different quality attributes concerning the network data delivery performance. The proposed architecture provides server application developers with an object-based communication library (similar to the standard TCP/IP socket library), that supports different classes of service. Implementation issues and optimization strategies are also discussed. Quantitative measures aimed at evaluating the effectiveness of the proposed architecture are provided.

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. A large fraction of users access network resources through web clients/browsers. As far as the Internet scenario is concerned, 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 [19]. Furthermore, 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. 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, 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.

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 [6]. It is thus desirable that network servers (e.g., Web, Video on Demand, and FTP servers) are 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 design and the implementation of an operating system extension for service differenti-

ation of communication-bound processes. The architecture we propose has been designed with an object-oriented approach. 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 is provided 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 in the TimeSys Linux/RT kernel [*www.timesys.com*], which also offers a complete support for POSIX threads.

The paper is organized as follows. A quick overview of the Linux real-time kernel we used is presented in section 2. Section 3 describes the proposed architectural model. We shall then concentrate, in section 4, on the implementation and optimization issues of the proposed 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 implementation, along with information on future work.

2. Linux Real Time

In order to obtain a complete control on system resources, we used a real-time operating system based on the Rate Monotonic Algorithm (RMA) [8]. The primary abstraction to resource management is the resource-set (`rset`) which represents a resource share controlled by real-time kernel. It allows applications to specify only their resource demands, leaving the scheduler to satisfy those demands using hidden resource management schemes. A resource-set is bound to one or more processes (or threads) by means of Process Identifiers (PIDs), and provides the exclusive use of its reserved amount of CPU. There are three parameters to specify `rset` behaviour: `T`, `C`, and `D`, where `T` represents a recurrence period, `C` represents the processing time required within `T`, and `D` is the deadline within which the `C` units of processing time must be available within `T`. According to the Rate Monotonic scheduling theory, a group of n independent periodic tasks can be shown to always meet their deadlines, providing the sum of the ratios `C/T` for each task is below an upper bound of overall CPU utilization. By doing this it is possible to make an admission control test for any new resource set. In particular we used a standard Linux Distribution (Suse 7.0 kernel 2.2.16) with a real-time module, provided by TimeSys, namely Linux/RT. This module works on CPU scheduling pipes provided by the operating system, by modifying the scheduling modules. In this manner the real-time module is completely transparent to those applications which do not use it explicitly. The scheduling algorithm used by Linux/RT is the Rate Monotonic Algorithm (RMA).

3. 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 the most tricky issue in order 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's amount depending on its service level. By doing this, 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 [13]. 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 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 the I/O activities from the CPU ones, by providing application developers with a communication library (cosSocket). Once separated these activities, I/O tasks will 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 [8]. 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 class is able to performs I/O operations with 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 library. The activities of cosManager, which are mainly I/O calls, are scheduled by the real-time kernel by means of the cosDaemon. In fact, such daemon is the only architecture's entity, capable of using real-time features in order to control all cosManagers belonging to the applications which reside on the end-

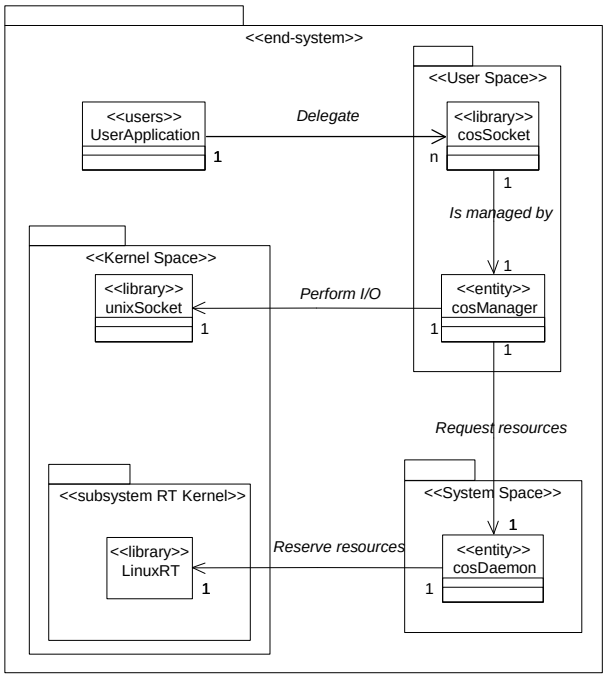

Figure 1. COMET Static Diagram of the proposed architecture.

system. As described later, the admission control policies for the guaranteed services are implemented by the cosDaemon. Furthermore, being the cosManager the responsible class for performing application I/O we were able to optimize the number of the system call issued to the kernel, as described in section 4.

As shown in figure 2, the cosSocket class acts as an interface between applications and the architecture. It makes available to the application developer a set of methods (similar to the standard socket library) that provide the necessary support for differentiated communication services.

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

3.1. Adaptive class service

By adaptive we mean a service class that can be requested without any admission control mechanism [20]. 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).

Let C be the total amount of CPU assigned to the Adaptive class service. We assume there are N_k cosSocket instances belonging to class k. The basic unit q of C to be assigned is calculated as:

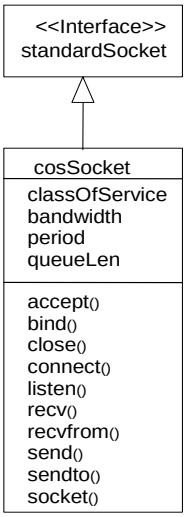

Figure 2. Class Diagram of cosSocket.

$$q = \frac{C}{\sum_{k=1}^{n} W_k N_k}$$

Therefore, each cosSocket instance will be assigned an amount of CPU equals to $W_k q$. The basic unit q is calculated each time an event occurs, that changes the classes' population, i.e. when a new instance is allocated, or an existing one is de-allocated. According to this scheme, the amount of CPU assigned to a certain class k, $W_k q$, is always greater than the quantity assigned to the class i, where $i < k$; thus the adaptive scheme guarantees that a higher service class will get a greater amount of CPU than a lower one. It is worth noting that it is possible to have $W_k N_k q < W_i N_i q$ where $i < k$. This is due to the fact that a lower class, i, has a greater number of instances. In practice, it is avoided that a reduced number of high priority instances is assigned a large amount of CPU, having a negative impact on lower priority ones. The adaptive service does not provide hard guarantee on the actual throughput of each process in every class of service; however, it allows to define several classes, providing a guaranteed differentiation between service classes on the basis of the servers' resources assigned to them.

3.2. 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. Such 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 faults).

4. Implementation issues

4.1. 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:

- *Synchronous model*. According to this model the communication between the cosSocket and cosManager is synchronous. No data copy is performed. Each application's I/O call corresponds to a system call issued by the cosManager in a synchronous way. The strength of this model is to avoid data copy.

- *Asynchronous model*. In this model, communication between the cosSocket and cosManager is asynchronous. In this case data to be sent are copied in an appropriate shared memory, managed by the cosManager. The implemented model is similar to the producer/consumer paradigm. Each application's I/O call corresponds to a system call issued by the cosManager in a asynchronous way, that is the system call can be deferred by the cosManager itself. The strength of this model is to decouple data production from data transfer.

- *Asynchronous Aggregated model*. This model is similar to the previous one, excepted from the fact that an optimized data management strategy is implemented. The idea is especially suited for TCP communications. Such strategy consists of reducing the number of system calls, issued by the cosManager, by aggregating data in several chunks. In practice, data sent by two or more application's I/O calls are aggregated in one chunk and then sent to the network by only one system call. The strength of this model is that of reducing the overhead due to the system calls.

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 are reported in section 5.

Figure 3 depicts the buffer management scheme with and without the aggregation.

From the figure, it is evident how in the first case (a) the cosManger performs 7 system calls (`send()`), while in the second case (b) the issued system calls are 3. We thus were able to significantly reduce the number of system calls. As

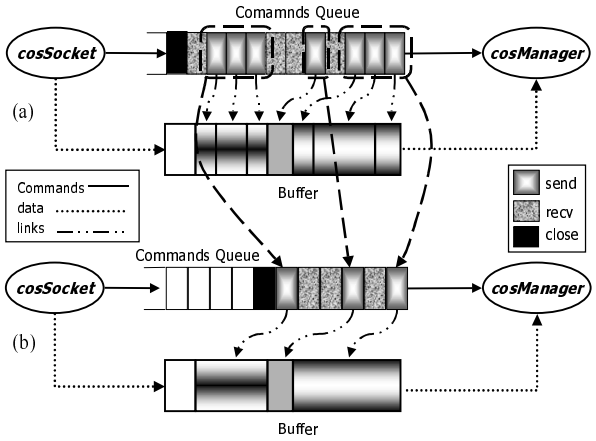

Figure 3. Asynchronous implementation model: (a) without aggregation; (b) with aggregation.

demonstrated in section 5, the aggregated implementation model effectively improves system's performance. All implementation models are based on the real-time scheduling mechanism controlled by cosDaemon. Each cosManager receives a CPU's amount depending from its class of service. When a cosManager is scheduled, it executes all I/O calls requested from application through cosSocket. The communication between cosManagers and the cosDaemon is realized by a Unix socket handled by the cosDaemon.

4.2. Setup algorithm

In order to manage both adaptive and guaranteed services, the cosDaemon needs to obtain a measure of the amount of CPU, namely `C`, necessary for saturating the throughput of external links. In fact, as far as I/O operations of data delivery servers are concerned, guaranteeing to them a CPU's amount greater than `C` is useless. Such CPU's amount is a percentage of the period, namely `T`. The value of `T` is fixed at 500 milliseconds. Such value is calculated as the average value of all possible ones. Therefore, at the startup time the system performs a setup procedure which aims to estimate the value of `C`. To the purpose, a benchmark procedure has been implemented, referred as "speed test". Such benchmark consists of a client and a server communicating by means of TCP protocol. Such benchmark measures the throughput at the server side. The implemented algorithm is presented in figure 4.

It is evident that in order to estimate the value of `C`, more than one measurement it is needed. The implemented algorithm converge to the minimum value of `C` which is necessary for obtaining the maximum throughput.

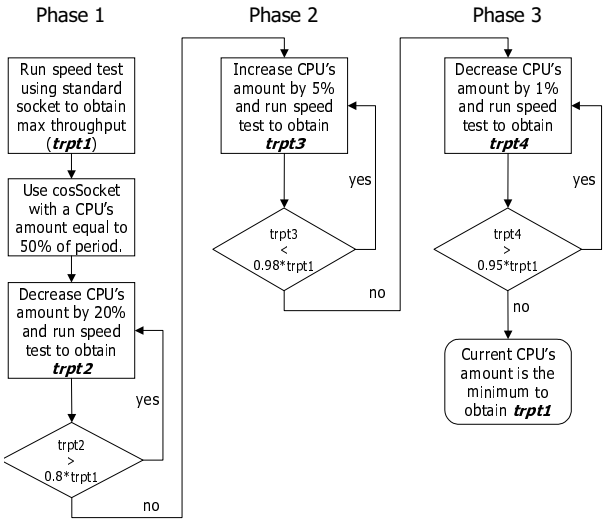

Figure 4. System setup algorithm.

4.3. Guaranteed service class

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 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 performance the following computation, diminishing the medium quadratic error: $u = u + k * e$ where k is a constant. The adopted scheme is illustrated in figure 5.

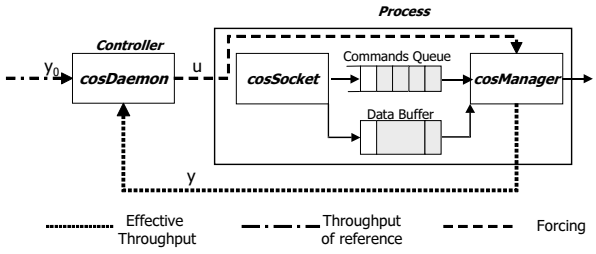

Figure 5. Self-regulating control loop for providing Guaranteed service.

5. Experimental results

In this section, we discuss the results obtained from the preliminary performance experiments we have conducted. The primary objectives of the experiments are:

1. To test and validate the different behaviour of the service classes;

2. To gain insight into the overhead introduced by the architecture;

3. To evaluate the impact of the different implementation models;

The testbed used is composed of a set of 3 commodity PCs (Pentium III 600Mhz with 256 Mbyte of RAM) wired by Fast-Ethernet switch. We test the architecture using applications communicating by means of the TCP protocol. During the tests, the external load was the normal background load of active services and applications. We did not mandate an idle workload because we felt the normal workload would be more representative of the workload that would be experienced in a cluster of non dedicated nodes.

Figure 6 shows the performance behaviour of the three implementation models.

As the figure shows, the asynchronous aggregated model has better performance than other ones. This is especially true when the application sends data using small packets (140 bytes), emphasizing the advantage of the adopted aggregation mechanism. Moreover, from figure 6 it is possible to evaluate the overhead introduced by the proposed architecture. In fact, an I/O-bound process (or thread) without cosSocket, requires about the 8% of CPU for saturating the external throughput (100 Mbit/s.), with packets of 2000 bytes. Using cosSocket, instead, the same experiment needs about 13% of CPU. The resulted overhead is thus about 5%. However, the architecture overhead strictly depends on the connection bandwidth between servers. At time of this writing, we tested the architecture only on a local network scenario (fast-ethernet) and in this case the needed percentage of CPU is always less the 15%. We are currently investigating the behaviour on a wide-area network with a cluster of servers connected with a Gigabit ethernet. As far as the guaranteed service class is concerned, as mentioned this kind of service was implemented by means of a self-regulating utilization control loop. In order to test the effectiveness of this mechanism we launched an FTP server, which uses the cosSocket with packets of 140 bytes, requesting a guaranteed service class with a value of throughput of 30 Mbit/s. Experimental results are reported in figure 7. It should be noted, that the system reaches the steady state after 13 intervals (we assumed that each intervals is 500 ms) with 8% of CPU utilization.

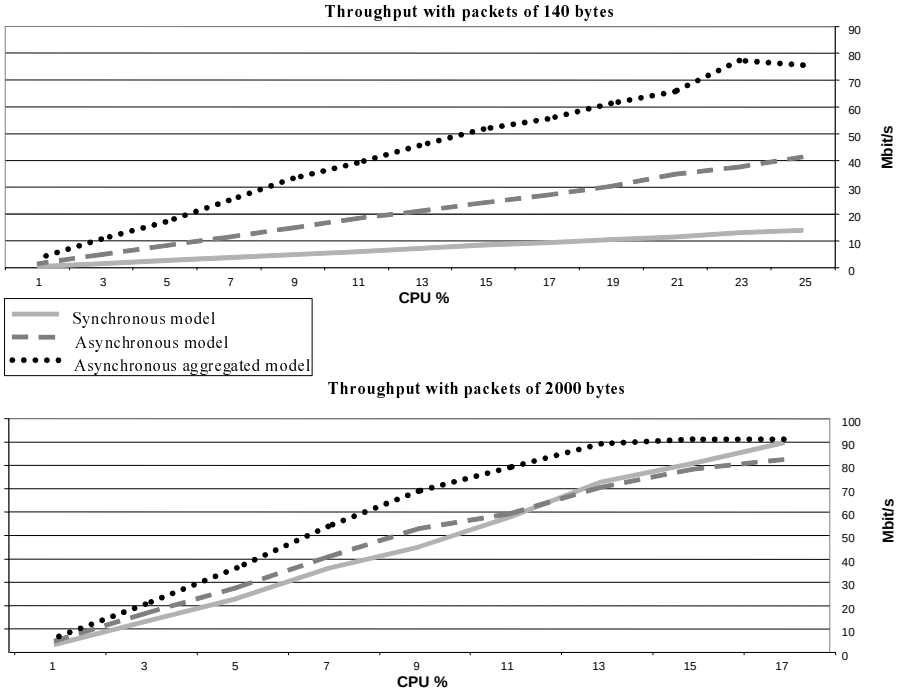

Figure 6. Impact on performance of the three implementation models.

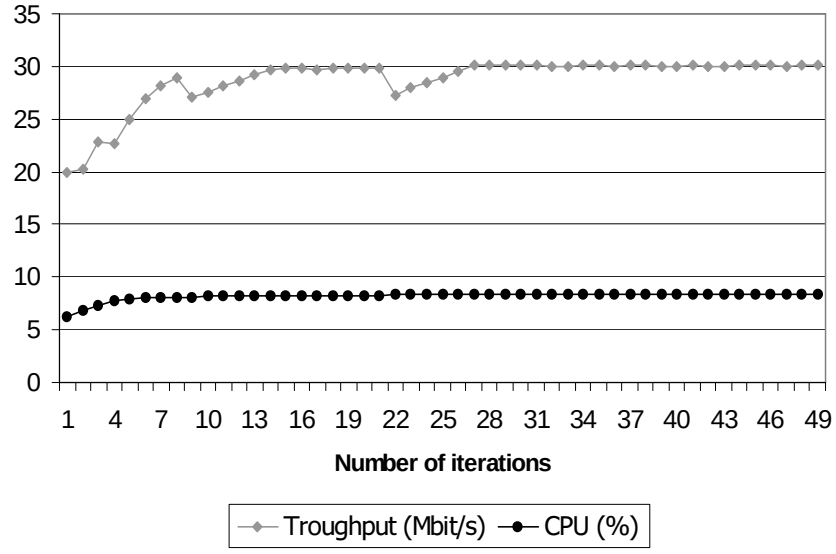

Figure 7. Guaranteed service class experiments.

As far as the latency is concerned, we measured the variance of the inter-arrival times between two applications (client and server) executed on two different hosts. We used both TCP and UDP protocol, but the results were not significantly different. Experimental results are presented in figure 8, where the server used TCP connections.

As the figure shows, we measured the variance (normalized at 1) under varying of CPU utilization and the period T. It is worth noting that in any cases the value of the variance is always less than 10^{-3}. Such value can be sufficient for a large part of real-time multimedia applications.

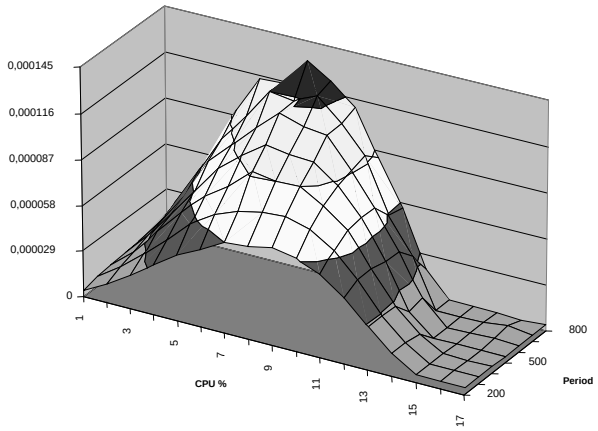

Figure 8. Variance of Inter-arrival times.

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 [3, 4] and Video on Demand [5, 6, 7] services. Many works like [11] and [12] as well as our previous experience in quality of service support [13], 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 [14] are proposed some architectural mechanisms to manage communication resources for guaranteed-QoS connections, and in [15, 21] 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. Many kernel extensions have been proposed to provide real-time guarantees for QoS-sensitive applications. For example, capacity reserves [16] have been used in Mach to allocate processing capacity for multimedia applications [17], and flexible CPU reservations was used in Rialto for efficient scheduling of time-constrained independent activities [18]. There are also different distributed processing schemes that allow end-to-end QoS support in middleware [9, 10]. In all revised works, resources control was used to increase performances 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 the design and implementation of an operating system extension for quality-of-service differentiation among classes of communication-bound processes. Our strategy relied on the capability of controlling the number of system calls issued for requesting an I/O task. We evaluated three different implementation models and we can state that for this kind of application an optimized buffer management is a tricky issue. The preliminary experimental results demonstrated the effectiveness of this approach. Future research activity will aim at refining measurements in order to demonstrate that proposed architecture effectively improves service availability in wide aerea scenario. In fact, we are currently investigating the behaviour on a wide-area network with a cluster of servers connected with a Gigabit ethernet. Finally, we are currently investigating the behaviour of the cosSocket when on the same end-system are executed applications which use standard socket.

Acknowledgements

The authors would like to acknowledge the excellent work done by Alfonso Sparano in code writing activity and Simon Pietro Romano for providing them excellent suggestions about the adoption of the Linux real-time operating system. This work has been performed under the financial support of Italian Ministry for University and Science and Technology Research (MURST) under grants “LABNET 2” and “MUSIQUE”.

References

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

[2] 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.

[3] T.F. Abdelzaher, K.G. Shin, “QoS Provisioning with qContracts in Web and Multimedia Servers”, in *Proc. of the 20th IEEE Real-Time Systems Symposium*, Aug. 2000.

[4] 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.

[5] F. Cao, J. Smith, K. Takahashi, “An Architecture of Distributed Media Servers for Supporting Guaranteed QoS and Media Indexing”, in *Proc. of the IEEE Conference on Multimedia Computing and Systems, Volume II (ICMCS’99)*, Florence, Italy, June 1999.

[6] K. Nahrstedt, H. DeMiguel, J. Liu, “QoS-aware resource management for distributed multimedia applications”, in *the Journal of High-Speed Networks, Special Issue on Multimedia Networking*, Giu. 1998.

[7] J. Park, S. Kim, J. Lee, S. Lee, “A Flexible Communication Architecture to Support Multimedia Services in High Speed Network”, in *Proc. of the 13th International conference on Information Networking (ICOIN’98)*, Apr. 2000.

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

[9] D. Le Tien, O. Villin, C. Bac, “Resource Managers for QoS in CORBA”, in *Proc. of the Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Apr. 2000.

[10] F. Kuhns, D. C. Schmidt, and D. L. Levine, “The Performance of a Real-Time I/O Subsystem for QoS-Enabled ORB Middleware”, in *Proc. of the International Symposium on Distributed Objects and Applications*, Set. 1999.

[11] A. Campbell, G. Coluson, D. Hutchison, “A continuous Media Transport and Orchestration Service”, in *Proc. of the ACM conference (SIGCOMM’92)*, Maryland (USA), 1992.

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

[13] 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.

[14] 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.

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

[16] C. Mercer, S. Savage, H. ToKuda, “Processor capacity reserves: Operating system for multimedia applications”, in *Proceedings of the IEEE Conference on Multimedia Computing and Systems*, May 1994.

[17] H. Tokuda, T. Nakajima, P. Rao,“Real Time Mach: Towards a predictable real-time system”, in *Proc. of the USENIX Mach Workshop*, pages 73-82, October 1990.

[18] M. Jones, D. Rosu, M. Rosu, “CPU reservation and time constraints: efficient, predictable scheduling of independent activities”, in *Proc. of the ACM Conference (SOSP97)*, October 1997.

[19] 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.

[20] 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.

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