Proceedings

Real-Time Systems Symposium

San Juan, Puerto Rico
December 7 – 9, 1994

Sponsored by
The IEEE Computer Society
Technical Committee on Real-Time Systems

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

Proceedings

Real-Time Systems Symposium

San Juan, Puerto Rico
December 7 – 9, 1994

Sponsored by
The IEEE Computer Society
Technical Committee on Real-Time Systems

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

IEEE Computer Society Press
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

IEEE Computer Society Press Order Number 6600-02
IEEE Catalog Number 94CH35728
ISBN 0-8186-6600-5 (paper)
ISBN 0-8186-6601-3 (microfiche)
ISBN 0-8186-6602-1 (case)
ISSN 1052-8725

Additional copies may be ordered from:

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264
Tel: +1-714-821-8380
Fax: +1-714-821-4641
Email: cs.books@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: +1-908-981-1393
Fax: +1-908-981-9667

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM
Tel: +32-2-770-2198
Fax: +32-2-770-8505

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
JAPAN
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553

Editorial production by Penny Storms
Cover by Joseph Daigle / Schenk-Daigle Studios
Printed in the United States of America by Braun-Brumfield, Inc.

The Institute of Electrical and Electronics Engineers, Inc.

Proceedings

Real-Time Systems Symposium

Message from the Conference Chairs

The Real-Time Systems Symposium provides an annual forum for exchanging information on emerging principles and practices underlying real-time computing. As in recent years, we continue to witness increasing interest in this field due to a better appreciation of the need for rigorous and scientific solutions to the highly interrelated problems of developing systems that have demanding requirements for correctness, dependability and timeliness. Many of the ideas formulated by researchers in the recent past are now being deployed in mainstream applications. This has given a major impetus to the field as a growing number of researchers and developers tackle the many challenging problems that remain. On a worldwide scale, RTSS is attracting a large international contingent: in the program committee, in terms of the submissions, and in the accepted papers. To encourage the dissemination of findings in experimental development work, we have continued the synopses sessions from the previous three years.

The technical program consists of 22 regular papers and 10 synopsis selected from a total of 107 submissions from fifteen countries. The program reflects recent developments in architecture, communication, databases, operating systems, performance, programming languages, scheduling, and formal methods for real-time applications. It also reflects an increased emphasis on system and tool implementation, evidencing a maturation of the underlying principles.

Each submission received at least four reviews. We are extremely grateful to the program committee members for their care and diligence in obtaining quality reviews and for making hard decisions during the program committee meeting. Outside reviewers did an excellent job in providing additional guidance, for which we express our sincere thanks.

RTSS '94 would not be possible but for the able and hard work of many. We wish to thank each member of the conference committee for helping to ensure that the symposium runs smoothly: Walt Heimerdinger for expertly handling of the conference finances, Sandra Ramos-Thuel who was instrumental in bringing RTSS '94 to San Juan and thereby taking RTSS outside the 48 contiguous states for the first time, Wei Zhao for his timely handling of all matters related to publicity, Prabha Gopinath for serving as our liaison with industry, and Linda Buss for her professional handling of registration. Finally, our thanks to Penny Storms of the IEEE Computer Society Press for her efforts related to the publication of the proceedings.

Farnam Jahanian
General Chair
University of Michigan

Krithi Ramamritham
Program Chair
University of Massachusetts, Amherst

Conference Committee

General Chair
Farnam Jahanian, *University of Michigan*

Program Chair
Krithi Ramamritham, *University of Massachusettes*

Treasurer
Walter Heimerdinger, *Honeywell*

Publicity
Wei Zhao, *Texas A&M University*

Industrial Chair
Prabha Gopinath, *Honeywell*

Local Arrangements
Sandra Ramos-Thuel, *AT&T*

Registration
Linda Buss

Program Committee

George Avrunin
U. of Massachusetts

Azer Bestavros
Boston U.

Alex Buchmann
Tech. Hochschule Demstadt

Alan Burns
U. of York

Flaviu Cristian
U. of Calif. at San Diego

Wolfgang Halang
Fern U.

Jayant Haritsa
Indian Institute of Science

Michelle Hugue
Trident Systems, Inc.

Kane Kim
U. of California at Irvine

Hermann Kopetz
Technical U. of Vienna

Christian Koza
Alcatel Austria

John Lehoczky
Carnegie Mellon U.

Jay Lala
C.S. Darper Labs. Inc

Jane W.S. Liu
U. of Illinois

Mirek Malek
U. of Texas at Austin

Al Mok
U. of Texas at Austin

Frank Olken
Lawrence Berkeley Lab

Raj Rajkumar
Software Engineering Inst.

Parmesh Ramanathan
U. of Wisconsin-Madison

Karsten Schwan
Georgia Inst. of Tech.

Alan Shaw
U. of Washington

Chia Shen
MERL

Kang Shin
U. of Michigan

Lorenzo Strigini
IEI-CNR

Jay Strosnider
Carnegie Mellon U.

Morikazu Takegaki
Mitsubishi Electric Corp.

Hide Tokuda
Carnegie Mellon U./Keio U.

Satish Tripathi
U. of Maryland

Fritz Vaandrager
CWI

Tetsuo Wasano
Nippon Telegraph & Tel Corp

Victor Yodaiken
New Mexico Inst. of Mining and Tech.

Reviewers

Parosh Aziz Abdulla
Ilsoo Ahn
George Avrunin
Parosh Aziz
Luiz Bacellar
Michael A. Barborak
Azer Bestavros
Pravin Bhagwat
Sushil Birla
Bard Bloom
Holger Branding
Guillaume P. Brat
Alex Buchmann
Wayne Burleson
Alan Burns
Saurav Chatterjee
B. Chen
Linda Christoff
Matthew Clegg
Flaviu Cristian
Stuart W. Daniel
Rob Davis
Wu Feng
Christoph Fetzer
Harold Forbes
Lars-ake Fredlund
Emmerich Fuchs
Alan Garvey
Kaushik Ghosh
Sunondo Ghosh
David Griffioen
Rhan Ha
Wolfgang Halang
Ching-Chih Han
Hans Hansson
Jayant Haritsa
Peter Holzer
S. Hong
Tai-Yi Huang
Michelle Hugue
David Hull
S. Hwang
Atri Indiresan
Farnam Jahanian
Kevin Jeffay
Kevin Kettler
Kane Kim
Yu-Seok Kim
Steven Klusener
Hermann Kopetz
Christian Koza
A. Krüger
James Kurose
Jay Lala
G. Leber
John Lehoczky
Kwei-Jay Lin
Jane Liu
Miroslaw Malek
Dietmar Millinger
Ichiro Mizunuma
A.K. Mok
Hien Nguyen
Douglas Niehaus
Frank Olken
Ard Overkamp
Mihir Pandya
Jaehyun Park
Ian Philp
M. Cristina Pinotti
Peter Puschner
Raj Rajkumar
Krithi Ramamritham
Parmesh Ramanathan
Jennifer Rexford
Hal Rosenberg
Debanjan Saha
John E. Sasinowski
Anton Schedl
Ralf Schlatterbeck
Karsten Schwan
Stephen Serafin
Lui Sha
Alan Shaw
Chia Shen
Kang Shin
Eltefaat H. Shokri
H. Shrikumar
Nandit Soparkar
Sang Son
John A. Stankovic
Alex Stoyenko
Lorenzo Strigini
Jay Strosnider
Doug Stuart
Chien-Chun Su
Chittur Subbaraman
Jun Sun
Arun Swaminathan
Hideyuki Takada
Morikazu Takegaki
Too-Seng Tia
Ken Tindell
Hide Tokuda
Satish Tripathi
Fritz Vaandrager
Gary Vondran
Alexander Vrchoticky
Fuxing Wang
Xin Wang
Tetsuo Wasano
Wang Yi
Victor Yodaiken
Myungchul Yoon
Wei Zhao
Shlomo Zilberstein
K.M. Zuberi

Table of Contents

Message from the Conference Chairs viii
Conference Committee ix
Reviewers x

Session 1: Scheduling and Resource Allocation I
Chair: Raj Rajkumar, SEI/CMU

Efficient Aperiodic Service under Earliest Deadline Scheduling 2
M. Spuri and G.C. Buttazzo

Mechanisms for Enhancing the Flexibility and Utility of Hard Real-Time Systems 12
N.C. Audsley, R.I. Davis, and A. Burns

Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Priority Systems Using Slack Stealing 22
S.R. Thuel and J.P. Lehoczky

Session 2: Databases and Resource Management
Chair: Sang Son, University of Virginia

Timeliness via Speculation for Real-Time Databases 36
A. Bestavros and S. Braoudakis

Resource Management for Continuous Multimedia Database Applications 46
J. Huang and D.-Z. Du

Session 3: Communications I
Chair: Chia Shen, MERL

Scaling and Performance of a Priority Packet Queue for Real-Time Applications 56
D. Picker and R.D. Fellman

A Priority Forwarding Router Chip for Real-Time Interconnection Networks 63
K. Toda, K. Nishida, E. Takahashi, and Y. Yamaguchi

Multiple Route Real-Time Channels in Packet-Switched Networks 74
K.C. Kwan and P. Ramanathan

Session 4: Compilers
Chair: Doug Locke, Loral Federal Systems

Busy-Idle Profiles and Compact Task Graphs: Compile-Time Support for Interleaved and Overlapped Scheduling of Real-Time Tasks 86
R. Gupta and M. Spezialetti

An Accurate Worst Case Timing Analysis for RISC Processors 97
S.-S. Lim, Y.H. Bae, G.T. Jang, B.-D. Rhee, S.L. Min, C.Y. Park, H. Shin, K. Park, and C.S. Kim

Compiler Transformations for Speculative Execution in a Real-Time System 109
M.F. Younis, T.J. Marlowe, and A.D. Stoyenko

Session 5: Formal Methods
Chair: Lui Sha, SEI/CMU

The Generalized Railroad Crossing: A Case Study in Formal Verification of Real-Time Systems 120
C. Heitmeyer and N. Lynch

Modeling and Analysis of Real-Time Ada Tasking Programs 132
J.C. Corbett

Response-Time Bounds of Rule-Based Programs under Rule Priority Structure 142
R.-H. Wang and A.K. Mok

Synopsis Session 6: Experimental Systems and Applications
Chair: Bhavani Thuraisingham, MITRE Corporation

A Solution to an Automotive Control System Benchmark 154
H. Kopetz

Applying RMA to Improve a High-Speed, Real Time Data Acquisition System 159
D. del Val and A. Viña

ARINC 659 Scheduling: Problem Definition 165
T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini

Session 7: Tools
Chair: Christian Koza, Alcatel Austria

Bounding Worst-Case Instruction Cache Performance 172
R. Arnold, F. Mueller, D.B. Whalley, and M. Harmon

Deterministic Upperbounds of the Worst-Case Execution Times of Cached Programs 182
J.-C. Liu and H.-J. Lee

Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes 192
R. Gerber, S. Hong, and M. Saksena

Session 8: Scheduling and Resource Allocation II
Chair: Michelle Hugue, Trident Systems, Inc.

Scheduling Adaptive Tasks in Real-Time Systems 206
K. Wang and T.-H. Lin

Dynamic End-to-End Guarantees in Distributed Real-Time Systems 216
M. Di Natale and J.A. Stankovic

On-Line Scheduling to Maximize Task Completions 228
S.K. Baruah, J. Haritsa, and N. Sharma

Session 9: Communications II
Chair: Satish Tripathi, University of Maryland

Probabilistic Bounds on Message Delivery for the Totem Single-Ring Protocol 238
L.E. Moser and P.M. Melliar-Smith

Real-Time Communication Services in a DQDB Network 249
R.L.R. Carmo, F. Vasques, and G. Juanole

Analysing Real-Time Communications: Controller Area Network (CAN) 259
K.W. Tindell, H. Hansson, and A.J. Wellings

Synopsis Session 10: Formal Approaches and Languages
Chair: Tom Lawrence, Rome Labs

Computing Quantitative Characteristics of Finite-State Real-Time Systems 266
S. Campos, E. Clarke, W. Marrero, M. Minea, and H. Hiraishi

Verifying an Intelligent Structural Control System: A Case Study 271
W.M. Elseaidy, R. Cleaveland, and J.W. Baugh, Jr.

Flexible Real-Time SQL Transactions 276
P.J. Fortier, V.F. Wolfe, and J.J. Prichard

Synopsis Session 11: Operating Systems and Communications
Chair: Jim Ready, MRI

Supporting Real-Time Traffic on Ethernet 282
C. Venkatramani and T.-c. Chiueh

Modeling DSP Operating Systems for Multimedia Applications 287
D.I. Katcher, K.A. Kettler, and J.K. Strosnider

Emulating Soft Real-Time Scheduling Using Traditional Operating System Schedulers 292
B. Adelberg, H. Garcia-Molina, and B. Kao

Author Index 299

Session 1: Scheduling and Resource Allocation I

Chair: Raj Rajkumar, SEI/CMU

Efficient Aperiodic Service under Earliest Deadline Scheduling

Marco Spuri* Giorgio C. Buttazzo

Scuola Superiore S.Anna
via Carducci, 40 - 56100 Pisa - Italy
spuri@pegasus.sssup.it, giorgio@sssup1.sssup.it

Abstract

In this paper we present four new on-line algorithms for servicing soft aperiodic requests in real-time systems, where a set of hard periodic tasks is scheduled using the Earliest Deadline First (EDF) algorithm. All the proposed solutions can achieve full processor utilization and enhance aperiodic responsiveness, still guaranteeing the execution of the periodic tasks. Operation of the algorithms, performance, schedulability analysis, and implementation complexity are discussed and compared with classical alternative solutions, such as background and polling service. Extensive simulations show that algorithms with contained run-time overhead present nearly optimal responsiveness.

A valuable contribution of this work is to provide the real-time system designer with a wide range of practical solutions which allow to balance efficiency against implementation complexity.

1 Introduction

Many complex control applications include tasks which have to be completed within strict time constraints, called deadlines. If meeting a given deadline is critical for the system operation, and may cause catastrophic consequences, that deadline is considered to be *hard*. If meeting time constraints is desirable, but missing a deadline does not cause any serious damage, then that deadline is considered to be *soft*. In addition to their criticalness, tasks that require regular activations are called *periodic*, whereas tasks which have irregular arrival times are called *aperiodic*.

The problem of scheduling a mixed set of hard periodic tasks and soft aperiodic tasks in a dynamic environment has been widely considered when periodic tasks are executed under the Rate Monotonic (RM) scheduling algorithm [11]. Lehoczky *et al.* [10] investigated server mechanisms (Deferrable Server and Priority Exchange) to enhance aperiodic responsiveness. Sprunt *et al.* [14] described a better service mechanism, called Sporadic Server (SS). Then, Lehoczky and Ramos-Thuel [8] found an optimal service method, called Slack Stealer, which is based on the idea of "stealing" all the possible processing time from the periodic tasks, without causing their deadlines to be missed. The same algorithm has been extended in [13] to handle hard aperiodic tasks, and in [6], to treat a more general class of scheduling problems.

All these methods assume that periodic tasks are scheduled by the RM algorithm. Although RM is an optimal algorithm, it is static and in the general case cannot achieve full processor utilization. In the worst case, the maximum processor utilization that can be achieved is about 69% [11], whereas in the average case, for a random task set, Lehoczky *et al.* [9] showed that it can be about 88%.

For certain applications requiring high processor workload, a 69% or an 88% utilization bound can represent a serious limitation. Processor utilization can be increased by using dynamic scheduling algorithms, such as the Earliest Deadline First (EDF) [11] or the Least Slack algorithm [12]. Both algorithms have been shown to be optimal and achieve full processor utilization, although EDF can run with smaller overhead.

Scheduling aperiodic tasks under the EDF algorithm has been investigated by Chetto and Chetto [4] and Chetto *et al.* [5]. These authors propose acceptance tests for guaranteeing single sporadic tasks, or group of precedence related aperiodic tasks. Although optimal from the processor utilization point of view, these acceptance tests present a quite large overhead to be practical in real-world applications.

Three server mechanisms under EDF have been recently proposed by Ghazalie and Baker in [7]. The authors describe a dynamic version of the known De-

*This work has been supported in part by the CNR of Italy under a research grant.

ferrable and Sporadic Servers [14], called Deadline Deferrable Server and Deadline Sporadic Server, respectively. Then, the latter is extended to obtain a simpler algorithm called Deadline Exchange Server.

The aim of our work is to provide more efficient algorithms for the joint scheduling of random soft aperiodic requests and hard periodic tasks under the EDF policy. Our proposal includes four algorithms having different implementation overheads and different performances. We first present an algorithm, called Dynamic Priority Exchange, which is an extension of previous work under Rate Monotonic (RM). Although much better than background and polling service, it does not offer the same improvement as the others. A completely new "bandwidth preserving algorithm", called Total Bandwidth Server, is also introduced. The algorithm significantly enhances the performance of the previous servers and can be easily implemented with very little overhead, thus showing the best performance/cost ratio. Finally, we present an optimal algorithm, the EDL Server, and a close approximation of it, the Improved Priority Exchange, which has much less run–time overhead. They are both based on off–line computations of the slack time of the periodic tasks. The proposed algorithms provide a useful framework to assist an HRT system designer in selecting the most appropriate method for his or her needs, by balancing efficiency with implementation overhead.

In the definition of our algorithms, we assume that all periodic tasks have hard deadlines coincident with the end of their periods, constant period T_i and constant worst case execution time C_i. All aperiodic tasks do not have deadlines and their arrival time is unknown.

For the sake of clarity, all properties of the proposed algorithms are proved under the above assumptions. However, they can easily be extended to handle periodic tasks whose deadlines differ from the end of the periods and that have non null phasing. In this case, the guarantee tests would only provide sufficient conditions for the feasibility of the schedule. Shared resources can also be included using the same approach found in [7], assuming an access protocol like the Stack Resource Policy [1] or the Dynamic Priority Ceiling [3]. The schedulability analysis would be consequently modified to take into account the blocking factors due to the mutually exclusive access to resources.

Due to lack of space, all proofs and some of the simulations have been omitted. See [15] for a complete description.

2 The Dynamic Priority Exchange Algorithm

In this section we introduce the Dynamic Priority Exchange server, DPE from now on. The main idea of the algorithm is to let the server trade its run–time with the run–time of lower priority periodic tasks (under EDF this means a longer deadline) in case there are no aperiodic requests pending. In this way, the server run–time is only exchanged with periodic tasks, but never wasted (unless there are idle times). It is simply preserved, even if at a lower priority, and it can be later reclaimed when aperiodic requests enter the system.

2.1 Definition of the DPE Server

The DPE server is an extension of the Priority Exchange server [10] adapted to work with the EDF algorithm. In the definition of the server we make use of *aperiodic capacities*, associated to the server itself and to each deadline of periodic task instances. They are updated by the algorithm we are going to describe and, when greater than zero, are considered by the scheduler as schedulable entities. When scheduled, they are used to service pending aperiodic requests.

The server has a specified period T_S and a capacity C_S. At the beginning of each period, the server's aperiodic capacity, C_S^d, where d is the deadline of the current server period, is set to C_S. Each deadline d associated to the instances (completed or not) of the i–th periodic task has an aperiodic capacity, $C_{S_i}^d$, initially set to 0. The aperiodic capacities (those greater than 0) receive priorities according to their deadlines and the EDF algorithm, like all the periodic task instances (ties are broken in favour of capacities, *i.e.*, aperiodics). Whenever the highest priority entity in the system is an aperiodic capacity of C units of time the following happens:

- if there are aperiodic requests in the system, these are served until they complete or the capacity is exhausted (each request consumes a capacity equal to its execution time);
- if there are no aperiodic requests pending, the periodic task having the shortest deadline is executed; a capacity equal to the length of the execution is added to the aperiodic capacity of the task deadline and is subtracted from C (*i.e.*, the deadlines of the highest priority capacity and the periodic task are exchanged);

- if neither aperiodic requests nor periodic task instances are pending, there is an idle time and the capacity C is consumed until, at most, it is exhausted.

In order to implement the algorithm, the only operations required in case of deadline exchange, are to update the values of two capacities and to check whether the "running" one is exhausted. Furthermore, the ready queue can be at most twice as long as without the server (there is at most one aperiodic capacity for each periodic task instance). From these simple observations we can conclude that whereas the implementation of a DPE server is not trivial, the run–time overhead does not significantly increase the typical overhead of a system using an EDF scheduler.

As far as the schedulability is concerned, the DPE server behaves like any other periodic task. The difference is that it can trade its run–time with the run–time of lower priority tasks. When a certain amount of time is traded, one or more lower priority tasks are run at a higher priority level, but their lower priority time is preserved for possible aperiodic requests. This run–time exchange does not affect the schedulability of the task set, as shown in the following Theorem.

Theorem 1 *Given a set of periodic tasks with processor utilization U_P and a DPE server with processor utilization U_S, the whole set is schedulable* if *and* only if

$$U_P + U_S \leq 1,$$

where U_P and U_S are the utilization factors of the periodic task set and the DPE server, respectively. □

2.2 Resource Reclaiming

In most typical real–time systems, the processor load of periodic activities, either statically or dynamically, is guaranteed *a–priori*. This means that the maximum possible load reachable by periodic tasks is taken into account. When this peak is not reached, that is, the actual execution times are lower than the worst case values, it is not always obvious how to reclaim the spare time for real–time activities (a trivial approach is to execute background tasks).

In a system with a DPE server is very simple to reclaim the spare time of periodic tasks for aperiodic requests. It is sufficient that when a periodic task completes, its spare time is added to the corresponding aperiodic capacity. An example of this behaviour is depicted in Figure 1. When the first aperiodic request enters the system at time $t = 4$, one unit of time is available with deadline 8, and three units are available

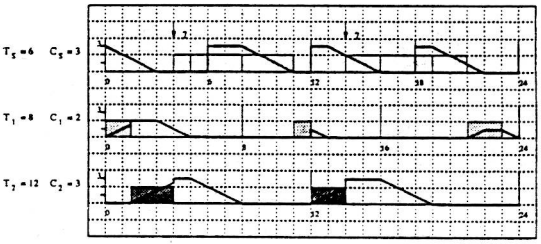

Figure 1: DPE server resource reclaiming.

with deadline 12. The aperiodic request can thus be serviced immediately for all the seven units of time required, as shown in the schedule.

Without the reclaiming described, at time $t = 4$ there would be a half unit of time available with deadline 8 and two and a half units available with deadline 12. The request would be serviced immediately for six units of time, but the last unit would be delayed until time $t = 11$, when it would be serviced in background (neither periodic tasks nor aperiodic capacities would be ready at that time).

Note that reclaiming the spare time of periodic tasks as aperiodic capacities does not affect the schedulability of the system. It is sufficient to observe that, when a periodic task has spare time, this time has been already "allocated" to a priority level corresponding to its deadline when the task set has been guaranteed. That is, the spare time can be safely used if requested with the same deadline. But this is exactly the same as adding it to the task corresponding aperiodic capacity.

3 The Total Bandwidth Algorithm

A different approach that we can follow to improve the aperiodic response times is to assign a possible short deadline to each aperiodic request. The assignment must be done in such a way that the overall processor utilization of the aperiodic load never exceeds a specified maximum value U_S.

This approach is the main idea behind the Total Bandwidth Server (TBS), which we define in the following section. The name of the server comes from the fact that, each time an aperiodic request enters the system, the total bandwidth of the server, whenever possible, is immediately assigned to it.

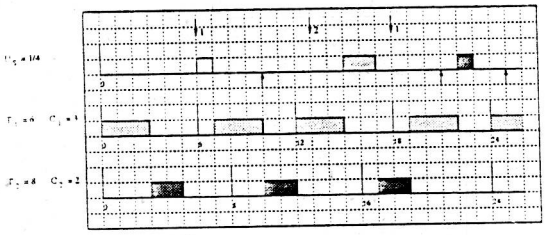

Figure 2: Total Bandwidth server example.

3.1 Definition of the TB Server

The definition of the TB server is very simple. When the k-th aperiodic request arrives at time $t = r_k$, it receives a deadline

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k}{U_S},$$

where C_k is the execution time of the request and U_S is the server utilization factor (*i.e.*, its bandwidth). By definition $d_0 = 0$. The request is then inserted into the ready queue of the system and scheduled by EDF, as any other periodic instance or aperiodic request already present in the system.

Note that we can keep track of the bandwidth already assigned to other requests by simply taking the maximum between r_k and d_{k-1}. Intuitively, as it is stated in Lemma 1, the assignment of the deadlines is such that in each interval of time the ratio allocated by EDF to the aperiodic requests never exceeds the server utilization U_S, that is, the processor utilization of the aperiodic tasks is at most U_S.

In Figure 2, an example of schedule produced by the TB server is depicted. The first aperiodic request, arrived at time $t = 6$, is serviced (*i.e.*, scheduled) with deadline $d_1 = r_1 + \frac{C_1}{U_S} = 6 + \frac{1}{0.25} = 10$. 10 being the earliest deadline in the system, the aperiodic activity is executed immediately. Similarly, the second request receives the deadline $d_2 = r_2 + \frac{C_2}{U_S} = 21$, but it is not serviced immediately, since at time $t = 13$ there is an active periodic task with a shorter deadline (18). Finally, the third aperiodic request, arrived at time $t = 18$, receives the deadline $d_3 = \max(r_3, d_2) + \frac{C_2}{U_S} = 21 + \frac{1}{0.25} = 25$ and is serviced at time $t = 22$.

To show that full processor utilization can be achieved with a TB server, too, we have first proved that the aperiodic processor utilization does not actually exceeds U_S.

Lemma 1 *In each interval of time $[t_1, t_2]$, if C_{ape} is the total execution time demanded by aperiodic requests arrived at t_1 or later and served with deadlines less than or equal to t_2, then*

$$C_{ape} \leq (t_2 - t_1)U_S.$$

□

Now the following Theorem holds.

Theorem 2 *Given a set of n periodic tasks with processor utilization U_P and a TB server with processor utilization U_S, the whole set is schedulable* if *and* only if

$$U_P + U_S \leq 1.$$

□

3.2 Implementation Complexity

The implementation of the TB server is the simplest among those seen so far. In order to correctly assign the deadline to the new issued request, we only need to keep track of the deadline assigned to the last aperiodic request (d_{k-1}). Then, the request can be queued into the ready queue and treated by EDF as any other periodic instance. Hence, the overhead is only due to the increased length of the ready queue if several aperiodic requests are pending at the same time. However, this problem can be solved by managing a separate FIFO queue for the aperiodic requests, and inserting only the first one into the ready queue. In this way the overall overhead is practically negligible.

4 The EDL Algorithm

The Total Bandwidth algorithm is able to achieve good aperiodic response times with extreme simplicity. Still we could desire a better performance if we agree to pay something more. For example, looking at the schedule in Figure 2, we could argue that the second and the third aperiodic requests may be served as soon as they arrive, without compromising the schedulability of the system. The reason for this is that, when the requests arrive, the active periodic instances have enough effective laxity (*i.e.*, the interval between the completion time and the deadline) to be safely preempted. The main idea of the EDL algorithm is to take advantage of these laxities.

4.1 Definition of the EDL Server

The definition of the EDL server makes use of some results presented by Chetto and Chetto in [4]. In this

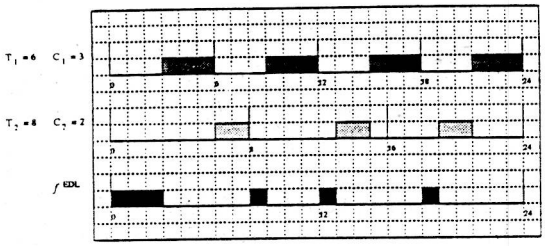

Figure 3: Availability function under EDL.

paper, two different implementations of EDF, namely EDS and EDL, are studied. Under EDS the active tasks are processed as soon as possible, while under EDL they are processed as late as possible. An accurate characterization of the idle times produced by the two algorithms is given. Moreover, a formal proof of the optimality, in the sense that it guarantees the maximum idle time in a given interval, is stated for EDL. In the original paper, this result is used to build an acceptance test for sporadic tasks (*i.e.*, aperiodics with hard deadlines) entering the system, while here it is used to build an optimal server mechanism for soft aperiodic activities.

Let us introduce the terminology used by the authors in [4]. With f_Y^X they denote the availability function

$$f_Y^X(t) = \begin{cases} 1 & \text{if the processor is idle at } t \\ 0 & \text{otherwise,} \end{cases}$$

defined with respect to a task set Y and a scheduling algorithm X. The function $f_{\mathcal{J}}^{\text{EDL}}$, with $\mathcal{J} = \{\tau_1, \tau_2\}$, is depicted in Figure 3. The integral of f_Y^X on an interval of time $[t_1, t_2]$ is denoted by $\Omega_Y^X(t_1, t_2)$: it gives the total idle time in the specified interval.

The result of optimality addressed above is stated in Theorem 2 of [4], which we recall here.

Theorem 3 *Let $\mathcal{A}$ be any aperiodic task set and X any preemptive scheduling algorithm. For any instant t,*

$$\Omega_{\mathcal{A}}^{\text{EDL}}(0,t) \geq \Omega_{\mathcal{A}}^{X}(0,t).$$

□

This result lets us build an optimal server using the idle times of an EDL scheduler. In particular, given the periodic task set, the function f_Y^X, which is periodic with *hiperperiod* $H = \text{lcm}(T_1, \ldots, T_n)$, can be represented by means of two vectors. The first, $\mathcal{E} = (e_0, e_1, \ldots, e_p)$, represents the times at which idle times occur, while the second, $\mathcal{D}^* = (\Delta_0^*, \Delta_1^*, \ldots, \Delta_p^*)$,

i	0	1	2	3
e_i	0	8	12	18
Δ_i^*	3	1	1	1

Figure 4: Idle times under EDL.

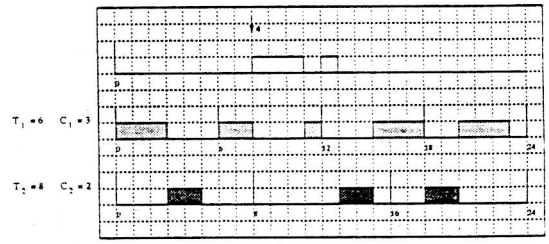

Figure 5: Example of schedule produced with an EDL server.

represents the lengths of these idle times. The two vectors for the example of Figure 3 are shown in Figure 4 (note that we can have idle times only after the arrival time of a periodic task instance).

The EDL server mechanism is based on the following idea: the idle times of an EDL scheduler are used to schedule aperiodic requests as soon as possible, postponing the execution of periodic activities, similarly to the effect of the "Slack Stealer" of [8]. The optimality stated in Theorem 3 will give us the optimality of the server built with this idea.

In particular, when there are no aperiodic activities in the system, the periodic tasks are scheduled according to the EDF algorithm. Whenever a new aperiodic request enters the system (and no previous aperiodic is still active) the set $\mathcal{J}(t)$ of the current active periodic tasks, plus the future periodic instances, is considered. The idle times of an EDL scheduler applied to $\mathcal{J}(t)$, that is, $f_{\mathcal{J}(t)}^{\text{EDL}}$, are then computed and consequently used to schedule the current aperiodic requests. See Figure 5 for an example. Note that the response time of the aperiodic request is optimal.

The procedure to recompute at each new arrival the idle times of EDL applied to $\mathcal{J}(t)$ is described in [4] and is not reported here. The worst case complexity of the algorithm, which is $O(Nn)$, where N is the number of distinct periodic requests that occur in $[0, H[$, and n is the number of periodic tasks, is relatively high and can give the algorithm little practical interest. As for the "Slack Stealer", the EDL server will be used to provide a lower bound to the aperiodic response times, and to build a nearly optimal implementable

algorithm, described in the next section.

4.2 EDL Server Properties

The analysis of the EDL server schedulability is quite straightforward. In fact, the server allocates to the aperiodic activities only the idle times of a particular EDF schedule, without compromising the timeliness of the periodic tasks. This is more precisely stated in the following Theorem.

Theorem 4 *Given a set of n periodic tasks with processor utilization U_P and the corresponding EDL server (the behaviour of the server strictly depends on the characteristics of the periodic task set), the whole set is schedulable* if *and* only if

$$U_P \leq 1$$

(the server automatically allocates the bandwidth $1 - U_P$ to aperiodic requests). □

The property of optimality addressed above, that is, the minimization of the response times of the aperiodic requests, is stated in the following Lemma.

Lemma 2 *Let X be any on-line preemptive algorithm, $\mathcal{J}$ a periodic task set, and J an aperiodic request. If $c^X_{\mathcal{J}\cup\{J\}}(J)$ is the completion time of J when $\mathcal{J}\cup\{J\}$ is scheduled by X, then*

$$c^{\text{EDL server}}_{\mathcal{J}\cup\{J\}}(J) \leq c^X_{\mathcal{J}\cup\{J\}}(J).$$

□

5 The Improved Priority Exchange Algorithm

Although optimal, the algorithm described in the previous section has too much overhead to be considered practical. However, its main idea can be usefully adopted to develop an implementable algorithm, still maintaining a nearly optimal behaviour, as shown later in the discussion of the simulations.

What makes the EDL server not practical is the complexity of computing the idle times at each new aperiodic arrival. This computation must be done each time in order to take into account the periodic instances partially executed or already completed at the time of arrival.

We can avoid the heavy idle time computation using the mechanism of priority exchanges. With this mechanism, in fact, the system can easily keep track

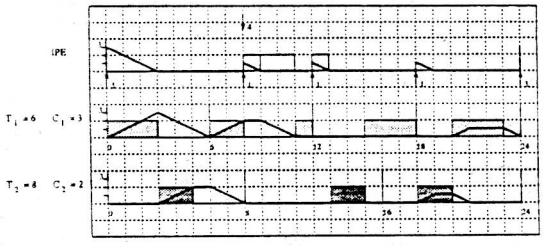

Figure 6: Improved Priority Exchange server example.

of the time advanced to periodic tasks and possibly reclaim it at the right priority level. The idle times of the EDL algorithm can be precomputed off–line. The server can use them to schedule aperiodic requests, when there are any, or to advance the execution of periodic tasks. In the latter case the idle time advanced can be saved as aperiodic capacity at the priority levels of the periodic tasks executed.

5.1 Definition of the IPE Server

To obtain the Improved Priority Exchange (IPE) algorithm, we modify the DPE server using the idle times of an EDL scheduler. First, we obtain a far more efficient replenishment policy for the server. Second, the resulting server is no longer periodic and it can always run at the highest priority in the system.

The IPE server is thus defined in the following way:

- the IPE server has an aperiodic capacity, initially set to 0;
- at each instant $t = e_i + kH$, with $0 \leq i \leq p$ and $k \geq 0$, a replenishment of Δ^*_i units of time is scheduled for the server capacity, that is, at time $t = e_0$ the server will receive Δ^*_0 units of time (the two vectors $\mathcal{E}$ and $\mathcal{D}^*$ have been defined in the previous section);
- the server priority is always the highest in the system, regardless of any other deadline;
- all other rules of IPE (aperiodic requests and periodic instances executions, exchange and consumption of capacities) are the same as for a DPE server.

The same task set of Figure 5 is scheduled with an IPE server in Figure 6. Note that the server replenishments are set according to the function $f^{\text{EDL}}_{\mathcal{J}}$, illustrated in Figure 3.

The IPE schedulability is stated in the following Theorem.