On Scheduling Imprecise Tasks in Real-Time Distributed Systems

Rômulo Silva de Oliveira

II - Univ. Fed. do Rio Grande do Sul
Caixa Postal 15064
Porto Alegre-RS, 91501-970, Brasil
romulo@inf.ufrgs.br

Joni da Silva Fraga

LCMI-DAS - Univ. Fed. de Santa Catarina
Caixa Postal 476
Florianópolis-SC, 88040-900, Brasil
fraga@lcmi.ufsc.br

Abstract

The Imprecise Computation technique has been proposed as an approach to the construction of real-time systems that are able to provide both guarantee and flexibility. This paper analyzes the use of Imprecise Computation in the scheduling of distributed real-time applications. Initially it is presented an approach to the scheduling of distributed imprecise tasks. Then we discuss the main problems associated with that goal and some possible solutions.

Keywords: Real-time systems, scheduling, imprecise computation, distributed systems.

1. Introduction

Computer systems used in real-time applications are submitted to timing requirements besides their functional aspects. In these systems, results should be correct not only from the logical point of view, but they should also be generated at the right moment. A missed deadline provokes a temporal failure that can be critical in some cases.

A basic problem found in the construction of real-time distributed systems is the allocation and scheduling of tasks on the available resources. Real-time systems require the simultaneous achievement of two fundamental goals [4]: to guarantee that the results will be produced at the desired moment and to build flexibility into the system so it can adapt itself to a dynamic environment and increase its utility.

There are scheduling solutions that assume a fixed group of tasks to be executed. These solutions reserve resources for the worst case scenario. Worst case scenario in this context means that all tasks will arrive with maximum frequency, will present their maximum execution time and will arrive at the worst possible moment, with normally means all tasks will arrive simultaneously.

Solutions based on worst case analyzes are able to guarantee that, in any task arrival scenario, all tasks will meet their time constraints. However, this approach results in systems that are not flexible and where resources are under-utilized. Examples of this approach are found in [32]. There are also scheduling solutions that do not provide an off-line guarantee for deadlines. Although resources are used fully and the resulting system is quite flexible, the lack of a previous guarantee for its temporal behavior makes this solution unfeasible for the class of applications with critical timing requirements. Examples can be found in [23].

The Imprecise Computation technique [19] divides each application task into a mandatory part and an optional part. The mandatory part is able to generate a minimum quality result that is necessary to maintain the system operating in a safe way. The optional part refines this result until it has maximum quality. A mandatory part is said to generate an imprecise result, while the result of the mandatory + optional parts is said to be precise. A task is called imprecise if it is possible to decompose it into mandatory and optional parts. This technique tries to achieve those two fundamental objectives mentioned before: flexibility and off-line guarantee. Examples of applications can be found in [11] and [15].

The general goal of this paper is to discuss how real-time applications that use Imprecise Computation concepts can be scheduled in distributed systems. In other words, to show how Imprecise Computation can be adapted to an environment where tasks execute in different processors and the communication among them is based on messages. An approach to this scheduling problem is presented in section 2. Section 3 discusses the problem of scheduling imprecise tasks in real-time distributed systems. Concluding remarks appear in section 4.

2. Scheduling Approach

This section describes our approach to the scheduling of imprecise tasks in distributed systems. Figure 1 shows the approach composed by four algorithms. It was presented before in [21]. Algorithm 1 is responsible for the allocation of tasks to processors; algorithm 2 is able to verify that mandatory parts will always be concluded before their respective deadline; algorithm 3 is used to select optional parts that, at a given moment, should be considered for execution; algorithm 4 determines if a given optional part can be executed without jeopardizing the execution of mandatory parts.

The first algorithm corresponds to the process of task allocation. The initial task set is broken into **h** subsets, where **h** is the number of processors. Since a successful allocation demands all mandatory parts being guaranteed, the allocation algorithm must use algorithm 2. It also tries to balance the load to increase the odds of an optional part being executed.

The second algorithm must verify that no deadline will be missed when only mandatory parts are executed. The problem is complicated by the fact that tasks in different processors may communicate through messages. Therefore, the execution schedule of a processor can be affected by delays due to messages that come from other processors.

The third algorithm evaluates at run-time if a given optional part should be considered for execution. The rationale behind this scheme is to implement an admission policy that discards optional parts whose importance is very low. Obviously, the minimum level of importance for an optional part to be admitted depends on the processor load at each instant.

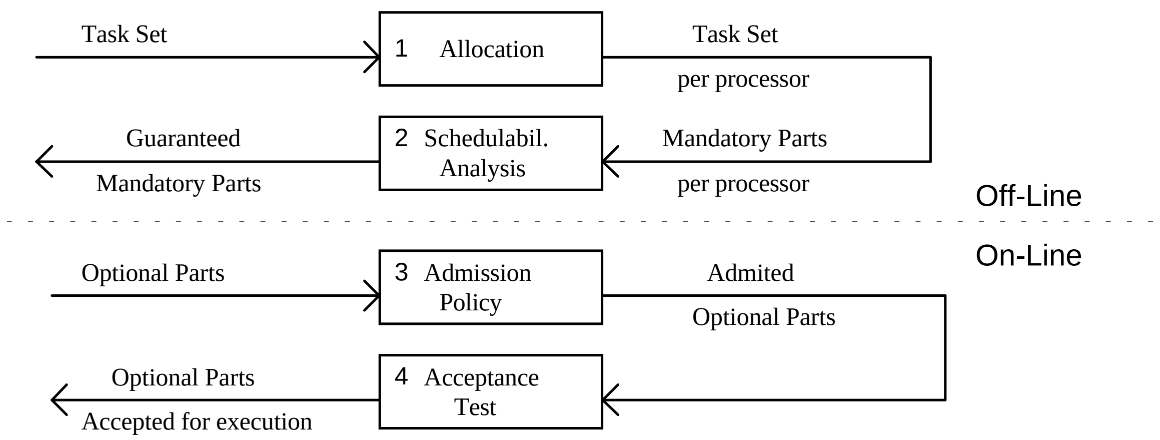

Figure 1 - General approach.

Fourth algorithm determines if a given optional part, previously admitted by algorithm 3, can be accepted for execution. An optional part can be accepted for execution only if it will not jeopardize previous guarantees. An optional part can be accepted if it does not alter the conditions supposed by algorithm 2 in such a way to invalidate the off-line schedulability test.

3. Discussion of the Problem

The problem of scheduling imprecise tasks in distributed real-time systems involves several aspects. The discussion of this problem is divided in four main aspects. The approach described in the previous section demands an algorithm for each one of the four specific problems.

3.1 Allocation

Real-time scheduling in a distributed context is generally solved in two stages. In the first stage tasks are allocated to processors. The second stage corresponds to the local scheduling of each individual processor. Migration of tasks is usually not allowed in real-time applications due to the cost of such operation, both in terms of resources and time. The fact that each task is allocated permanently to a processor increases the importance of an appropriate allocation.

The primary goal of allocation is to guarantee that all tasks can always conclude its respective mandatory part before deadline. In order to accomplish that, an allocation algorithm proposes several allocation solutions. For each one the algorithm uses a schedulability test to verify the schedulability of mandatory parts. With regard to the primary objective, any allocation that allows an off-line guarantee for the mandatory part of every task is equally satisfactory.

Allocation algorithms should also try, as secondary objective, to increase the chances of all optional parts to execute. Consider two allocation solutions X and Y, both equally satisfactory from the point of view of the primary objective. That is, mandatory parts can be guaranteed so much in X as in Y. It is possible that allocation X is better balanced with regard to allocation Y, so that optional parts will be more executed in X than in Y. Therefore, from the point of view of the secondary objective, allocation X is better than allocation Y.

A simple way to increase the odds of an optional part executing would be to balance system load taking into account the maximum execution time of each task. This basic approach can be improved. It is more realistic for the sake of load balancing to consider the average execution time of each task instead of worst-case execution time. Sporadic tasks are treated in worst-case analysis as periodic tasks with period equal to the minimum interval between activations. This treatment is too pessimistic for load balance analysis. It is more realistic to treat sporadic tasks as periodic tasks with period equal to the average time interval between activations. If each task has a fixed nominal value it is possible to use these nominal values to judge the load balance quality. Two situations are undesirable: to have too many optional parts in the same processor and to have very valuable tasks concentrated on the same processor. This analysis is not possible in applications where the nominal value of a task varies along the time.

Solutions in the literature tend to be sub-optimal, due to the problem complexity. It is possible to attack the allocation problem with heuristic searches. Certain problems allow the construction of good heuristics that are capable of arriving quickly at a good solution. When the allocation problem becomes more complex, it is difficult to create heuristics that capture all the tradeoffs and implications of the decision makings. Then alternative methods of global optimization, such as simulated annealing ([5],[16],[30]) and genetic algorithms [14], must be considered.

Simulated annealing requires the definition of an energy function that evaluates the quality of a given solution. This function must generate an energy value **E** such that a low energy is associated with a good solution, while a high energy is associated with a bad solution.The primary objective of allocation is to guarantee task deadlines when we consider only mandatory parts. The secondary objective will be to provide a reasonable load balance to avoid localized overloads and to increase the odds of optional parts to execute. We define this energy as:

$E = K_e \cdot E_e + K_b \cdot E_b$; where E_e and E_b represent the energy associated with task schedulability and load balance, respectively.

Values K_e and K_b are so that the aspect schedulability receives greater importance than the aspect load balance. A solution with good load balance but not schedulable will always have a smaller energy than a solution with terrible load balance but with all deadlines guaranteed. Values K_e and K_b will guarantee that schedulability is the primary allocation objective and load balance is just a secondary objective. The allocation algorithm looks for the solution with the best load balance among all solutions that guarantee system schedulability.

3.2 Guarantee for Mandatory Parts

The imprecise computation approach requires an off-line guarantee that each mandatory part will be finished before its respective deadline. In order to analyze this aspect of the problem it is possible to ignore the optional parts and to consider each task formed exclusively by mandatory part. The schedulability test goal is to guarantee mandatory parts. Optional parts will be executed with the remaining resources after this guarantee has been fulfilled.

There are in the literature countless scheduling solutions to guarantee deadlines when a group of tasks execute in a single processor. Similar solutions for distributed systems exist in a very smaller number than for the single processor case. Deterministic predictability for mandatory parts can be obtained, among other forms, through scheduling based on fixed priorities [18]. In this approach, tasks receive priorities according to some specific policy and a schedulability test is executed off-line. It determines if exist the guarantee that all tasks will be executed before deadline. The schedulability test must be compatible with the priority assignment policy. An on-line preemptive scheduler produces the execution schedule using the priorities previously assigned. Theory of fixed priority scheduling went through some important developments in recent years. It now supports task models that are much more complex ([2], [25]) than those supported by early works ([17], [18]).

It is common the appearance of precedence relationships among tasks in distributed systems. A precedence relationship is created by a need of synchronization and/or transmission of data between two tasks of the application. A message creates dependence between predecessor and successor tasks. The successor task can only begin its execution after receiving the message. Although this also happens in centralized systems, the fact that tasks are distributed among different processors hinders system scheduling. Any schedulability analysis for distributed systems should be able to work with precedence relations.

The work in [1] shows that it is possible to use offsets to implement precedence relations among tasks. By establishing an offset between the release of two tasks it is possible to guarantee that the successor will only begin its execution after the predecessor has concluded. This offset must be larger than or equal to the maximum response time of the predecessor task. In case data is transmitted from one task to the other, when the successor task is released after some time offset, it is guaranteed that the data is already available to be used as input. This technique is sometimes called static release of tasks [27], because the relative release time of tasks is previously defined in terms of offsets. The original system is transformed in an equivalent system where tasks are independent but present offsets that enforce the precedence relations. Schedulability tests are applied to this equivalent system.

Most published work use fixed priorities and static task release to implement precedence relations. Arbitrary precedence relations in monoprocessors are considered in [1] and [12]. Paper [31] is about arbitrary precedence relations in distributed systems. Works [25] and [27] deal with linear precedence, where each task has at most one predecessor and one successor.

It is also possible to implement precedence relations by an explicit message from the predecessor task to the successor task. This message informs that the predecessor task is finished and the successor task can be released. The message can also contain some data. This technique is sometimes called dynamic release of tasks [27], because the instant the successor task is released depends on the instant the predecessor task finishes and this value is only known at run-time. The uncertainty regarding the instant the successor task is released can be modeled as a release jitter [31]. For the sake of analyze, the original system is transformed in an equivalent system where tasks are independent but they present a release jitter. Again, schedulability tests are applied to the equivalent system.

Dynamic task release is used in [13] to implement linear precedence relations in monoprocessors. Dynamic release is also used in [28] and [31] in distributed systems analysis where there are only linear precedence relations. The only two works known by the authors that uses dynamic task release to implement arbitrary precedence relations in distributed systems are presented in [5] and [22].

3.3 Identification of Spare Capacity

In order to off-line guarantee that all mandatory parts will be concluded before their respective deadline, it is necessary to reserve resources for the worst-case scenario. Worst-case scenario, as used here, refers so much to execution times as to the worst possible combination of task releases. This scenario is tipically much more pessimist than the average case. Since resources were reserved for worst-case, every time a task exhibits a less demanding behavior it generates some spare capacity. There are several on-line factors capable of generating spare capacity with respect to system resources. This spare capacity is used to execute optional parts. The following paragraphs describe the many sources of spare capacity.

The processor time that remains free after reserving resources for the worst case of all mandatory parts is called the system remaining capacity. It can be dedicated to the execution of optional parts without jeopardizing the execution of mandatory parts. The remaining capacity is known off-line.

The probability of the worst-case scenario becoming true at run-time is very small. Most of the time, mandatory parts can be concluded before deadline with fewer resources than it was reserved off-line. This happens when a task activation occupies less processor time than the worst case foreseen at design time. The processor time that was reserved off-line but not used at run-time is called gain time. Its value and instant is only known at run-time.

It is also possible that a given sporadic task is activated with a smaller frequency than the maximum frequency established at design time. The mandatory part of this task will consume fewer resources than in the worst case. This processor time reserved but not used due a smaller activation frequency can also be used for execution of optional parts. Its value and instant of appearance is only known on-line.

Schedulability tests applied off-line always consider the worst possible combination of task releases. For example, tests based on the concept of the critical instant reserve resources for when all tasks are simultaneously released. Whenever tasks are not simultaneously released there will be a surplus of resources on-line. This happens frequently due to sporadic tasks, to periodic tasks with different periods, to release jitter and to precedence relations. All these situations generate free resources that can be used for execution of optional parts.

The sum of all these sources of resource results in spare capacity. This spare capacity should be detected and used for execution of optional parts. Algorithms for the detection of spare capacity tend to be simple and not very efficient or efficient but very complex. Also, algorithms that solve it must be used on-line, when they dispute resources (processor time) with the optional parts themselves.

An optimal acceptance test is defined as capable of detecting the whole spare capacity in the system as soon as it is generated. Optimal solutions for the acceptance test are unfeasible in practice, due to the complexity of its algorithms.

Many works in the literature try to give an on-line guarantee to aperiodic tasks. Although such algorithms have not specifically been created for imprecise computation, they can be used as acceptance tests for optional parts. That is possible because an optional part can be considered as an aperiodic task that was not guaranteed off-line and needs a dynamic guarantee.

In [24] there are algorithms capable of offering a dynamic guarantee for aperiodic tasks through explicit manipulation of the execution schedule. They maintain a table to keep track of when each task will occupy the processor. The table is built at run-time. This kind algorithm is called planning based.

The work in [26] presents servers that are capable of giving on-line guarantee to aperiodic tasks when fixed priorities are used. The servers reserve for themselves the remaining capacity in the system after all periodic tasks have been guaranteed. During the execution, the servers use this capacity to execute aperiodic tasks.

Static slack stealing is presented in [29] as a method to identify spare capacity when fixed priority is used. It builds a table off-line that indicates when the system can accept an aperiodic task without compromising tasks previously guaranteed. The table is kept updated at run-time.

Dynamic slack stealing was presented in [7]. It adapts the static slack stealing to a task model that includes synchronization among tasks, release jitter and guaranteed sporadic tasks. It also makes use of any processor time resulting from a task that does not use all the capacity reserved for it. This algorithm calculates the system spare capacity at run-time. The dynamic slack stealing algorithm results in excellent efficiency concerning the identification of spare capacity, but its overhead is very high and its use not feasible in practice.

In [3] and [8] it is proposed a solution based on the algorithm of dynamic slack stealing, called dynamic approximate slack stealing (DASS). This approach is such that every spare capacity identified really exists. However, this algorithm presents an efficiency inferior to the previous one, because it is not capable of identifying all the spare capacity existent at a given moment. Simulations presented in [8] show that, although it is not an optimal solution, DASS results in an efficiency that is bigger than solutions based on servers.

3.4 Selection of Optional Parts for Execution

Sometimes the spare capacity identified is not big enough to execute all optional parts available. In this case it is necessary to choose which optional parts available for execution in a certain moment should be executed and which should be discarded. This choice must consider the importance of each task. System overhead is also reduced because the acceptance test does not have to consider optional parts that were refused by the admission policy.

In many systems the importance of executing a certain task depends on the past behavior of the system. A common situation is periodic tasks where the importance of a precise execution increases when that task has been imprecisely executed in past activations. In this type of task the imprecision is somewhat cumulative. It is desirable to execute the task precisely to interrupt its accumulation of imprecision. This type of dependence is called intra-task [20]. Examples of intra-task dependence appear in applications such like radar systems and control systems [6].

Tasks with a producer-consumer relationship also presents dependence. A producer task generates as output data that will be used as input by the consumer task. Many algorithms present a smaller execution time when they receive an input data of better quality. So, by precisely executing a producer task we will decrease the execution time of the consumer task. This "extra processor time" generated can be used to execute other optional parts and increase system utility. This type of dependence is called inter-task [20]. Examples of tasks with inter-task dependence appear in applications such as voice recognition, radar systems and image processing for autonomous navigation [10].

Most imprecise computation studies consider that the value of a task is defined at arrival time. It is supposed that it is not affected by the behavior of other tasks or by what happened to previous releases of this same task. The works [6] and [10] constitute an exception. In [10] the task model includes inter-task dependence. Error in the input data of a task can extend the worst-case execution time of its own mandatory and/or optional parts. In [6] the task model includes intra-task dependence. It is supposed that all tasks should be precisely executed at least once every certain number of releases. In the two mentioned works the error model is such that an optional part could eventually become mandatory or the global mandatory execution time of the system could be increased.

Two heuristics, FCFS and AVDT, were described in [9] as admission policies for systems with optional components. When policy FCFS is used, all optional components are always admitted. When policy AVDT is used, only optional components with a value density greater than the average system value density are considered. The value density is obtained by the division of the task value by its worst-case execution time. In [20] two heuristics are proposed as admission policy for imprecise tasks. The two policies, CVDT and INTER, consider the existence of intra-task and inter-task dependence.

4. Conclusions

In this work we analyzed problems related with real-time scheduling of imprecise tasks in distributed systems. The approach was described, that is, how to schedule imprecise tasks in distributed systems. The problem analysis was divided in four aspects: allocation of tasks to processors, off-line guarantee for mandatory parts, on-line identification of spare capacity and on-line selection of optional parts for execution when it is not possible to execute everyone.

The approach proposed in this work simultaneously provides the necessary off-line guarantee for critical systems and increases system utility through execution of optional parts. The cost of a critical system is reduced because only the critical functions and properties receive a guarantee for its behavior in a worst-case situation. Scheduling of optional parts is based on best-effort. The Imprecise Computation technique can be seen as a mechanism capable of tolerating overloads without compromising its critical timing constraints. In this case, we have a graceful degradation when system quality is reduced. The contribution of this paper should be understood as the presentation of a complete approach for scheduling imprecise tasks in distributed systems and the discussion of some problems associated with that goal.

References

[1] N. Audsley, K. Tindell, A. Burns. The End of the Line for Static Cyclic Scheduling? Proceedings of the Fifth Euromicro Workshop on Real-Time Systems, IEEE Computer Society Press, pp. 36-41, june 1993.

[2] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, A. J. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. Software Engineering Journal, Vol. 8, No. 5, pp.284-292, 1993.

[3] N. C. Audsley, R. I. Davis, A. Burns. Mechanisms for Enhancing the Flexibility and Utility of Hard Real-Time Systems. Proceedings of the IEEE Real-Time Systems Symposium, pp. 12-21, San Juan, Puerto Rico, december 1994.

[4] A. Burns, A. J. Wellings. Criticality and Utility in the Next Generation. The Journal of Real-Time Systems, Vol. 3, correspondence, pp. 351-354, 1991.

[5] A. Burns, M. Nicholson, K. Tindell, N. Zhang. Allocating and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System. Proceedings of the Workshop on Parallel and Distributed Real-Time Systems, pp. 11-20, 1993.

[6] J. Y. Chung, J. W. S. Liu, K. J. Lin. Scheduling Periodic Jobs that Allow Imprecise Results. IEEE Trans. on Computers, Vol.39, No.9, pp.1156-1174, sep 1990.

[7] R. I. Davis, K. W. Tindell, A. Burns. Scheduling Slack Time in Fixed Priority Pre-emptive Systems. Proc. IEEE Real-Time Syst. Symp., pp.222-231, 1993.

[8] R. I. Davis. Approximate Slack Stealing Algorithms for Fixed Priority Pre-emptive Systems. Department of Computer Science report, Univ. of York, 1993.

[9] R. Davis, S. Punnekkat, N. Audsley, A. Burns. Flexible Scheduling for Adaptable Real-Time Systems. Proceedings of the IEEE Real-Time Technology and Applications Symposium, pp. 230-239, may 1995.

[10] W. Feng, J. W.-S. Liu. Algorithms for Scheduling Tasks with Input Error and End-to-End deadlines. Un. of Illinois at Urbana-Champaign, DCS-TR #1888, 1994.

[11] W. Feng, J. W.-S. Liu. Performance of a Congestion Control Scheme on an ATM Switch. Proc. of the Int. Conference on Networks, Florida, jan 1996.

[12] R. Gerber, S. Hong, M. Saksena. Guaranteeing End-to-End Timing Constraints by Calibrating Intermediate Processes. Proceedings of the IEEE Real-Time Systems Symposium, december 1994.

[13] M. G. Harbour, M. H. Klein, J. P. Lehoczky. Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems. IEEE Transactions on Software Engineering, Vol. 20, No. 1, pp. 13-28, january 1994.

[14] E. S. H. Hou, H. Ren, N. Ansari. Efficient Multiprocessor Scheduling Based on Genetic Algorithms. In "Dynamic, Genetic, and Chaotic Programming", Ed. B. Soucek, John Wiley & Sons, 1992.

[15] X. Huang, A. Cheng. Applying Imprecise Algorithms to Real-Time Image and Video Transmission. Proc. IEEE Workshop on Real-Time App. Chicago, may 1995.

[16] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. Optimization by Simulated Annealing. Science (220), pp. 671-680, 1983.

[17] J. Y. T. Leung, J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. Perf. Evaluation, 2(4), pp.237-250, dec. 1982.

[18] C. L. Liu, J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, Vol. 20, No.1, pp.46-61, jan 1973.

[19] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, J.-Y. Chung. Imprecise Computations. Proc. of the IEEE, Vol. 82, No. 1, pp. 83-94, january 1994.

[20] R. S. Oliveira, J. S. Fraga. Scheduling Imprecise Computation Tasks with Intra-Task / Inter-Task Dependence. Proceedings of the 21st IFAC/IFIP Workshop on Real Time Programming, Gramado-RS-Brasil, pp. 51-56, november 1996.

[21] R. S. Oliveira. Escalonamento de Tarefas Imprecisas em Sistemas Distribuídos. Tese de Doutorado, Dep. de Eng. Elét., Univ. Fed. Santa Catarina, fevereiro 1997.

[22] R. S. Oliveira. J. S. Fraga. Escalonamento de Tarefas com Relações Arbitrárias de Precedência em Sistemas Tempo Real Distribuídos. Anais do 16o Simpósio Brasileiro de Redes de Computadores, maio 1998 (english version available as technical report).

[23] K. Ramamritham, J. A. Stankovic, W. Zhao. Distributed Scheduling of Tasks with Deadlines and Resource Requirements. IEEE Transactions on Computers, Vol. 38, No. 8, pp. 1110-1123, august 1989.

[24] K. Ramamritham, J. A. Stankovic, P.-F. Shiah. Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 1, No. 2, pp. 184-194, april 1990.

[25] L. Sha, R. Rajkumar, S. S. Sathaye. Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems. Proceedings of the IEEE, Vol. 82, No. 1, pp. 68-82, january 1994.

[26] B. Sprunt, L. Sha, J. Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. The Journal of Real-Time Systems, Vol. 1, pp. 27-60, 1989.

[27] J. Sun, J. W.-S. Liu. Bounding the End-to-End Response Time in Multiprocessor Real-Time Systems. Proc. Workshop on Parallel and Distributed Real-Time Systems, pp.91-98, Santa Barbara, CA, april 1995.

[28] J. Sun, J. W.-S. Liu. Synchronization Protocols in Distributed Real-Time Systems. Proc. of 16th Int. Conference on Distributed Computing Systems, may 1996.

[29] S. R. Ramos-Thuel, J. P. Lehoczky. Algorithms for Scheduling Hard Aperiodic Tasks in Fixed-Priority Systems Using Slack Stealing. Proceedings of the 15th IEEE Real-Time Systems Symposium, pp.22-33, December 1994.

[30] K. W. Tindell, A. Burns, A. J. Wellings. Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy. Real-Time Syst., Vol.4, No.2, jun 1992.

[31] K. Tindell, J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. Microprocessors and Microprogramming, 40, 1994.

[32] J. Xu, D. L. Parnas. On Satisfying Timing Constraints in Hard-Real-Time Systems. IEEE Trans. on Software Engineering, Vol.19, No.1, pp.70-84, Jan 1993.