2001

Intelligent Feedback Control-based Adaptive Resource Management for Asynchronous, Decentralized Real-time Systems

B. Ravindran
Virginia Polytechnic Institute and State University

Pushkin Kachroo
University of Nevada, Las Vegas, pushkin@unlv.edu

T. Hegazy

Intelligent Feedback Control-Based Adaptive Resource Management for Asynchronous, Decentralized Real-Time Systems

Binoy Ravindran, Pushkin Kachroo, and Tamir Hegazy

***Abstract*—We present intelligent feedback control techniques for adaptive resource management in asynchronous, decentralized real-time systems. We propose adaptive resource management techniques that are based on feedback control theory and are designed using the intelligent control design paradigm. The controllers solve resource allocation problems that arise during run-time adaptation using the classic proportional integral derivative control functions and fuzzy logic. We study the performance of the controllers through simulation. The simulation results indicate that the controllers produce low missed deadline ratios and resource utilizations during situations of high workloads.**

***Index Terms*—Adaptive resource management, asynchronous decentralized systems, feedback control, fuzzy-logic, real-time systems.**

I. Introduction

Real-time computer systems that are emerging for the purpose of strategic mission management such as coordination of multiple entities that are manufacturing a vehicle, repairing a damaged reactor, or conducting combat are subject to great uncertainties at the mission and system levels. The computations in the system are predominantly asynchronous (mutually and globally), e.g., event driven and aperiodic, data-dependent, e.g., input data arrivals that cause significant changes in resource needs based on semantic content of data, and they frequently constitute an overload. Examples of asynchronous, decentralized real-time systems and applications include the emerging generation of surface combatant systems of the U.S. Navy (e.g., the Aegis) and the U.S. Army (e.g., the Patriot). Such real-time mission management applications require decentralization because of the physical distribution of application resources and for achieving survivability in the sense of continued availability of application functionality that is situation-specific. Due to their physical dispersal, most real-time distributed computing systems are "loosely" coupled using communication paradigms that employ links, buses, rings, etc., resulting in additional uncertainties—variable communication latencies, regardless of the bandwidth [1].

Most of the past efforts on real-time scheduling and resource management focus on synchronous, device-level, sampled data monitoring and regulatory control that is usually centralized but occasionally distributed [2]–[8]. The fundamental premise of the hard real-time computing theory is that the behavior of the application and the system can be made to be deterministic through extensive *a-priori* knowledge about load parameters, communications, exceptions, dependencies, and conflicts. The theory exploits such *a-priori* information and provides guarantees about application and system behavior under a set of tightly constrained mission and resource conditions that are anticipated in advance. Therefore, it is very difficult to practically employ, adapt, or scale such techniques for real-time systems that are decentralized and asynchronous [1], [9]–[11]. Asynchronous real-time computer systems and their applications are inherently *posteriori* in terms of their workload characteristics and thus require adaptive real-time resource management.

Manuscript received July 17, 2000; revised February 2, 2001. This work was supported by the U.S. Office of Naval Research under Grant N00014-00-1-0549. This paper was recommended by Associate Editor T. H. Lee.

The authors are with the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA (e-mail: binoy@vt.edu; pushkin@vt.edu; thegazy@vt.edu).

Publisher Item Identifier S 1094-6977(01)07597-6.

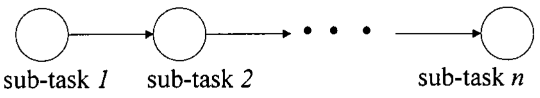

Fig. 1. Sequential subtasks of an end-to-end task.

In this paper, we propose adaptive resource management for asynchronous decentralized real-time systems that is based on feedback control theory. We propose feedback control functions that are based on the intelligent control design paradigm for achieving the application real-time requirements. The controllers perform adaptive resource management through run-time monitoring of application timeliness, feedback, and adaptation by application scaling. The controllers solve resource allocation problems such as determining the optimal number of replicas for load sharing using the classic proportional integral derivative (PID) control function. Further, we study controllers that perform resource allocation using fuzzy logic. The performance of the controllers is evaluated through simulations and using metrics such as missed deadline ratios and resource utilizations. The simulation results indicate that the controllers are very effective (in terms of the metrics) during situations of high workloads.

The rest of the paper is organized as follows. We describe the adaptive resource management problem that we are studying in Section II. Sections III and IV discuss PID feedback control techniques and fuzzy logic-based techniques for adaptive resource management, respectively. The experimental evaluation of the techniques is presented in Section V. Finally, the paper concludes with a summary of the work in Section VI.

II. Adaptive Resource Management Problem

To illustrate how feedback control laws can be constructed for performing adaptive resource management in asynchronous, decentralized real-time systems, we consider an example resource management problem. We use the example problem as a benchmark problem throughout the paper for designing feedback control techniques using different models.

We assume a distributed system with a transnode real-time task that is required to process data that arrives periodically. The upper bound on the size of the data that arrives during each period is assumed to be unknown *a-priori*. However, the task is required to complete each of its periods within a specified end-to-end deadline. The task is assumed to consist of n subtasks. The connectivity of the subtasks is assumed to be "sequential" i.e., subtask i needs to be completed before subtask $i+1$ can begin its execution (see Fig. 1). The subtasks of the task are assumed to be replicable so that the replicas can be dynamically executed on different computing nodes to exploit concurrency and achieve load sharing. Thus, replication is used as a means to reduce end-to-end task latencies when the data size increases at run-time and causes unacceptable task timeliness. The application hardware is assumed to consist of a set of homogenous processors that are distributed over a geographical area. The processors are interconnected together using a shared communication medium such as Ethernet (IEEE 802.3).

In designing a feedback controller for the benchmark problem, we define a three-fold objective.

1) Reduce the task execution time during overloaded situations that are caused due to high data stream sizes so that the task deadline can be satisfied.
2) Keep the processor and network utilization as low as possible.
3) Use the minimum number of subtask replicas.

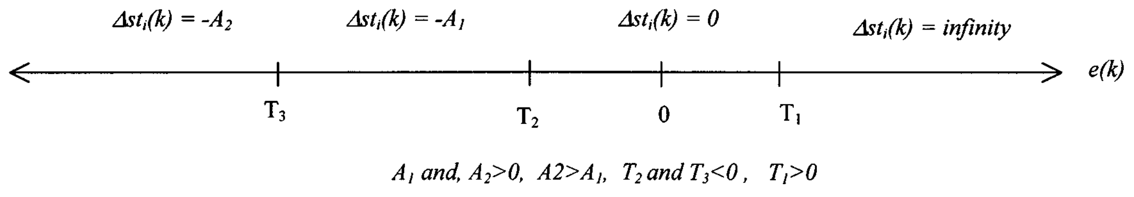

Fig. 2. Expert controller for adaptive resource management.

As an increase in the number of subtask replicas will reduce the task execution time but will increase the processor and network utilization, the controller has to compromise between the objectives so that the deadline can be satisfied with the minimum number of replicas and minimum resource utilizations. The controller for the problem, therefore, has to make the following decision. How many replicas of each subtask are needed for each period or what should be the change in the number of subtask replicas for each period?

III. Feedback Control

A. PID-Controller for Adaptive Resource Management

We design a PID control function for the benchmark adaptive resource management problem. The function uses the sum of weighted error, integral of error, and derivative of error terms as the control variable. To design a PID controller, we first define the sampling time. We define the sampling time to be the end of each period of data arrival. A decision is taken at the beginning of each period based on the behavior of the system in the past period(s). The controller input (i.e., the error) $e(k)$ can be defined as

$$\begin{aligned} e(k) = w_1(x(k) - X(k)) + w_2(u(k) - U(k)) \\ + w_3 n(k) + w_4 m(k) \end{aligned} \quad (1)$$

where

k	sampling instant;
$x(k)$	actual execution time of the task for processing the data that arrived in the previous period;
$X(k)$	desired execution time for the task;
$u(k)$	average actual utilization of the processors during the previous period;
$U(k)$	desired utilization;
$n(k)$	actual average actual network utilization throughout the previous period;
$m(k)$	missed deadline ratio.

Based on the error term, a PID control function that computes the change in the number of replicas for each subtask of the task is given by

$$\Delta \mathrm{st}_i(k) = k_{1i} e(k) + k_{2i} \sum_{j=0}^{k} e(j) + k_{3i}(e(k) - e(k-1)) \qquad i \in [0, 1, \ldots n]. \quad (2)$$

B. Expert Controller for Adaptive Resource Management

We also a design an expert controller that determines the change in the number of subtask replicas using the error described in (1). The expert control function characterizes the error values into four levels. For each level, the controller will perform a different decision as shown in Fig. 2.

The idea behind the function is summarized as follows.

1) If the error value exceeds a specified positive threshold T_1, then the performance of the task in the last period is assumed to be worse than the desired performance. This is probably because a large data stream was received in the last period and could not be satisfied by the current number of subtask replicas. Furthermore, most likely, the system is going to encounter a larger data stream in the current period. So the controller will make the maximum possible positive change in the number of subtask replicas to enable the system to handle the large data size.
2) If the error value falls between T_1 (which is positive) and T_2 (which is negative), then the task performance is assumed to be satisfactory and no change in the number of subtask replicas is required.
3) If the error value falls between T_2 and T_3 (which are both negative), the task performance is assumed to be much better than the desired performance. This is because the system is running a large number of subtask replicas, and therefore, the deadline could be possibly satisfied with less number of subtask replicas. Therefore, the controller will reduce the number of subtask replicas by a constant factor A_1.
4) If the error value is less than T_3, then the assumption made in step 3 holds. However, we will assume that the number of subtask replicas is much more than what is needed. So, the number of subtask replicas will be reduced by another constant factor A_2 that is larger than A_1

.

IV. Fuzzy Feedback Control

We also design control functions for the benchmark problem using fuzzy logic. To study the effect of choice of rules and inputs on controller performance, we consider two different types of fuzzy controllers, called Fuzzy1 and Fuzzy2. We use the singleton fuzzification process and centroidal method for defuzzification in designing the controllers. We also design a third controller called Fuzzy-Combined, that is designed using all the rules of the first two controllers as well as all their inputs. The controllers are described in the sections that follow.

A. Fuzzy1 Controller

The Fuzzy1 controller is based on the input fuzzy variables:

1) task execution time;
2) CPU utilization of the most loaded host;
3) network utilization;
4) history of missed deadline ratios.

The output of the controller is the desired change in subtask number of replicas. We use the membership function shown in Fig. 3 for all input fuzzy variables. As all the variables are always nonnegative, negative linguistic values are not considered.

Table I shows the values of a and b and the abbreviation used for each of the variables. Note that all values shown in Table I are empirical and are based on the fact that there is enough knowledge about

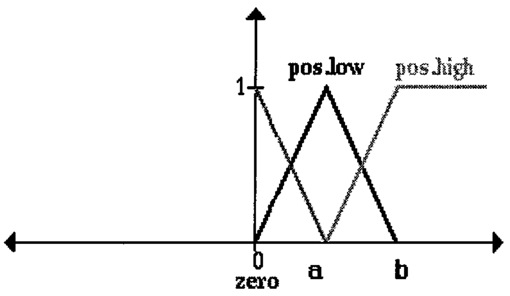

Fig. 3. Membership function for the input fuzzy variables.

TABLE I
VALUES OF FUZZY VARIABLES

Variable	Abbrv.	*a*	*b*
Task execution time	E	0.35 × relative deadline	0.7 × relative deadline
CPU utilization of most loaded host	P	25%	50%
Network utilization	N	35%	70%
Missed deadlines ratio history	M	40%	80%

the system under control. The output variable follows the membership function shown in Fig. 4, where the values of a, b, c, d, e, and f are $1, 3, 5, -1, -2$, and -4, respectively.

The Fuzzy1 controller has nine rules. We summarize the rules 1 through 7 in Table II. Rules 8, 9, and 10 are defined as rule 8) *IF* $N = \mathrm{PH}$ *THEN* $D = \mathrm{NL}$, rule 9) *IF* $M = \mathrm{PL}$ *THEN* $D = \mathrm{PL}$, and rule 10) *IF* $M = \mathrm{PH}$ *THEN* $D = \mathrm{PM}$, respectively, where D is the individual output of each fuzzy rule.

B. Fuzzy2 Controller

The Fuzzy2 controller is based on the input variables, error (ER) and change in the error (ERR), where ER is the same error in the paper submitted to OSDI, and ERR is the difference between the error value in the current sampling instance and the value in the previous sampling instance. The variable indicates how the error will be in the next sampling instance. The membership function of ER is shown in Fig. 5. The ERR membership function has only three linguistic values:

2) negative (N);
3) zero (Z);
4) positive (P).

The Fuzzy2 controller has 15 rules. The rules are summarized in Table III.

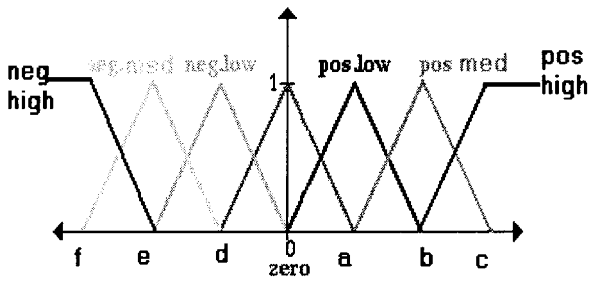

Fig. 4. Membership function for the output fuzzy variables.

TABLE II
FUZZY INFERENCE ENGINE RULES

		Variable *P*		
		Z (zero)	**PL (positive low)**	**PH (positive high)**
Variable *E*	**Z (zero)**	NM (negative medium)	NL (negative low)	
	PL (positive low)	NL (negative low)	Z (zero)	PL (positive low)
	PH (positive high)		PL	PM

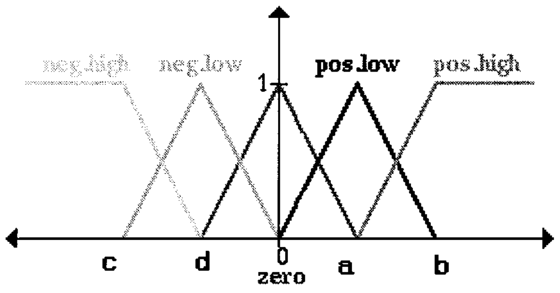

Fig. 5. Membership function of ER.

TABLE III
FUZZY INFERENCE ENGINE RULES

		Variable *ERR*		
		N	**Z**	**P**
Variable *ER*	**NH**	NH	NM	NL
	NL	NM	NL	Z
	Z	NL	Z	PL
	PL	Z	PL	PM
	PH	PL	PM	PH

V. EVALUATION OF FEEDBACK CONTROL TECHNIQUES

We evaluated the performance of the feedback control algorithms through a simulation study. The performance of the controllers were studied by comparing two "static" controllers, i.e., control functions that use a constant number of subtask replicas. The two static controllers used in the study include 1) a controller that uses the maximum number of subtask replicas called Static-6 and 2) a controller that uses half the maximum number of subtask replicas called Static-3. We use an increasing ramp pattern as the data stream size load scenario for the simulation experiments. Furthermore, we derive the parameters of the simulation environment from a real-time benchmark application that has resulted from our prior work [12].

We first simulated the performance of the nonfuzzy and fuzzy controllers separately and observed the following.

2) The PID and expert controllers outperformed the two static controllers in terms of reducing the missed deadline ratio and using the minimum number of subtask replicas simultaneously. Furthermore, between the PID and expert controllers, the PID controller performed marginally better than the expert controller. Also, the Static-6 controller was found to give the lowest possible missed deadline ratio among PID, expert, and Static-3 controllers. However, this controller uses a large number of replicas (more than what is necessary), and therefore, its CPU and network utilization is high. On the other hand, both the feedback control functions gave a nonoptimal but very low missed deadline ratio (compared to the static controllers) with a lower number of replicas. This is true for all the data values chosen for simulation.
3) Among the fuzzy controllers, Fuzzy1 was found to perform better than Fuzzy2 in terms of reducing the missed deadline ratio and the average CPU utilization during low data stream size situations. However, during high data stream size situations, Fuzzy2 outperformed Fuzzy1 in terms of reducing the average CPU utilization, average network utilization, and average number of subtask replicas. We also observed that Fuzzy-Combined, which is merely a union of all the rules in the inference engines of Fuzzy1 and Fuzzy2 controllers gave an overall balanced performance.

From these observations, we selected PID, Fuzzy-Combined, and the Static-6 controller for an "across the class" comparison. Figs. 6–9 show

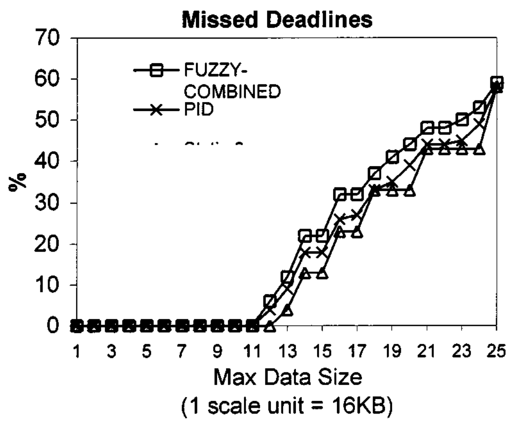

Fig. 6. Missed deadline percentages.

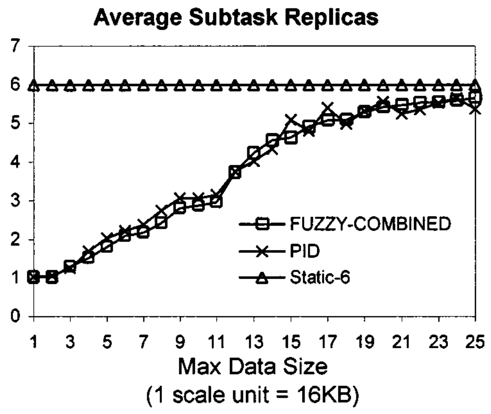

Fig. 7. Average number of subtask replicas.

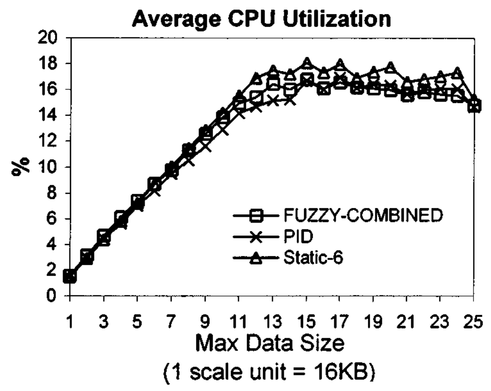

Fig. 8. Average CPU utilization.

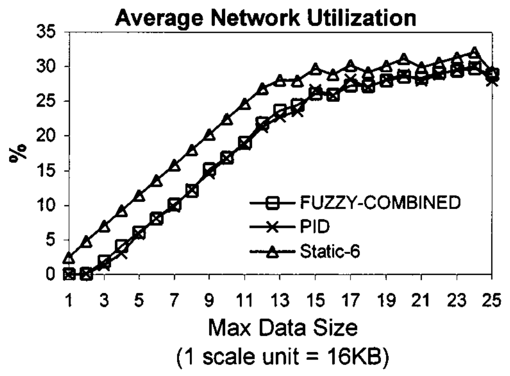

Fig. 9. Average network utilization.

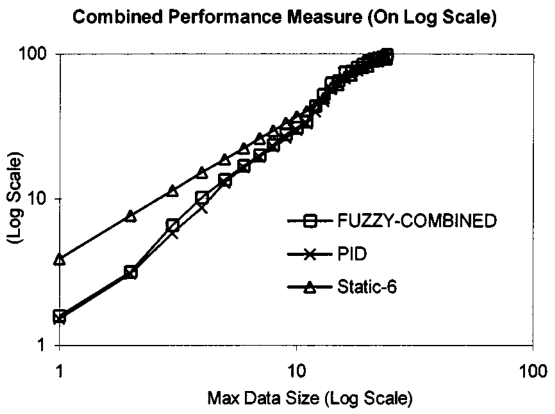

Fig. 10. Combined performance measure of fuzzy and nonfuzzy controllers.

the missed deadline percentages, average number of subtask replicas, average CPU utilizations, and average network utilization for these three controllers, respectively, as the maximum data stream size varies. We also show a combined performance measure of the three controllers in Fig. 10. The combined performance measure aggregates the three metrics of interest, namely, missed deadline percentages, average CPU utilizations, and average network utilization. For this aggregate metric, the lower the value, the better the performance of the algorithm. Thus, from the figures, we observe that the PID and the Fuzzy-Combined controllers perform almost the same for all data stream size situations (Fig. 10). Further, the two controllers clearly outperform the Static-6 controller. This indicates the promise of the PID and the Fuzzy controllers for adaptive resource management problems.

We note that the Fuzzy controller is easier to design than the PID controller. To design PID controllers that guarantee a desired transient and steady-state behavior, we need an analytical model of the system so that the control laws can be derived analytically. This is nontrivial for asynchronous decentralized real-time systems as they are inherently nonlinear. Traditional control designs do not provide a framework that facilitates this. On the other hand, the fuzzy controller can be designed based on the experience of the designer. (Note that we have also designed the PID controller here based on the control experience of the designer). However, they do not provide an analytically guaranteed behavior.

VI. Conclusion

In this paper, we proposed feedback controllers for an example asynchronous decentralized real-time system. The simulation results using the controllers showed superior performance of the PID and fuzzy controllers when compared to static controllers. This indicates the promise of feedback controllers for these types of problems. Thus, the contribution of the paper is the formulation of an example adaptive resource management problem for asynchronous decentralized real-time system—determining the number of subtask replicas and processors for executing them that will adapt the application at run-time to a given workload situation and will satisfy the real-time requirements—in the context of feedback control theory. Further, we show solutions to the problem using PID control and fuzzy control. Furthermore, we demonstrate the effectiveness of the solutions through a combination of benchmarking and simulation.

REFERENCES

[1] E. D. Jensen, “Asynchronous decentralized real-time computer systems,” in *Real-Time Computing, Proceedings of the NATO Advanced Study Institute*, W. A. Halang and A. D. Stoyenko, Eds. New York: Springer-Verlag, Oct. 1992.
[2] L. P. Briand and D. M. Roy, *Meeting Deadlines in Hard Real-Time Systems: The Rate Monotonic Approach*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1999.
[3] H. Kopetz, *Real-Time Systems: Design Principles for Distributed, Embedded Applications*. Boston, MA: Kluwer, 1997.
[4] S. H. Son, Ed., *Advances in Real-Time Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[5] J. A. Stankovic and K. Ramamritham, Eds., *Tutorial on Hard Real-Time Systems*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1988.
[6] ——, *Advances in Real-Time Systems*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1993.
[7] J. A. Stankovic *et al.*, *Deadline Scheduling for Real-Time Systems*. Boston, MA: Kluwer, 1998.
[8] A. M. Tilborg and G. M. Koob, Eds., *Foundations of Real-Time Computing: Scheduling and Resource Management*. Boston, MA: Kluwer, 1991.
[9] G. Koob, “Quorum,” in *Proc. Darpa ITO General PI Meet.*, Oct. 1996, pp. A-59–A-87.
[10] J. A. Stankovic *et al.*, “Strategic directions in real-time and embedded systems,” *ACM Comput. Surveys*, vol. 28, no. 4, pp. 751–763, Dec. 1996.
[11] D. B. Stewart and P. K. Khosla, “Mechanisms for detecting and handling timing errors,” *Commun. ACM*, vol. 40, no. 1, pp. 87–93, Jan. 1997.
[12] L. R. Welch and B. Shirazi, “A dynamic real-time benchmark for assessment of QoS and resource management technology,” in *Proc. Fifth IEEE Real-Time Technol. Applicat. Symp.*, June 1999.