Feedback Control Resource Management Using *A Posteriori* Workload Characterizations

David R. Alexander, Douglas A. Lawrence, and Lonnie R. Welch

School of Electrical Engineering and Computer Science
Ohio University
Athens, Ohio 45701
E-mail: {david.alexander|dal|welch}@ohio.edu

Abstract

Certain real-time applications must operate in highly dynamic environments, thereby precluding accurate characterization of the applications' workloads by static models. Thus, guarantees of real-time performance based on a priori characterizations are not possible. However, potential benefits of a posteriori approaches are significant, including the ability to function correctly in dynamic environments (through adaptability to unforeseen conditions), and higher actual utilization of computing resources.

In this paper, we consider a control theoretic framework that is appropriate for systems which experience large variations in workload. The goal is to manage a distributed collection of computing resources by continuously computing and assessing QoS and resource utilization metrics that are determined a posteriori. Potential benefits of developing a control theoretic framework for resource management include: reuse of the large body of results in control theory, focus on problem characteristics instead of control infrastructures, and stability analysis.

1 Motivation

The majority of real-time computing research has focused on the scheduling and analysis of real-time systems whose timing properties and execution behavior are known *a priori*. This is not without justification, since static approaches to the engineering of real-time systems have utility in many application domains [14]. Furthermore, the pre-deployment guarantee afforded by such approaches is highly desirable. However, there are numerous applications which must operate in highly dynamic environments, thereby precluding *accurate* characterization of the applications' properties by static models. In such contexts, temporal and execution characteristics can only be known *a posteriori*. Thus, guarantees of real-time performance based on *a priori* characterizations are extraneous. However, the potential benefits of *a posteriori* approaches are significant. These benefits include the ability to function correctly in dynamic environments (through adaptability to unforeseen conditions), and higher *actual* utilization of computing resources.

This paper deals with large, distributed real-time systems that have execution times and resource utilizations which cannot be characterized *a priori*. The motivation for our work is provided in part by the characteristics of *combat systems*, which are described in [7] as follows:

> *"The combat system processing demand per unit of time is defined as follows. Each tactical capability, e.g., track management, has its own processing demand. This demand is dependent on the number of objects, e.g., tracks, that will utilize this capability. The total number of capabilities active concurrently varies with time. The total number of objects driving each capability varies with time. Thus, the combat system processing demand per time unit is dependent on number of objects per capability in the time unit and the number of capabilities active during the time unit. …"*

Implications of these requirements are that demand space workload characterizations may need to be determined *a posteriori*, and an adaptive approach to resource allocation may be necessary. In existing real-time computing models, the execution time of a "job" is often used to characterize workload, and is usually considered to be known *a priori*. Typically, execution time is assumed to be an integer "worst-case" execution time (WCET), as in [11, 13, 23, 22, 21, 14, 3]. While [14] establishes the utility of WCET-based approaches by listing their domains of successful application, others [10, 8, 6, 9, 19, 13, 20, 18, 17, 12, 1, 2, 4] cite the drawbacks, and in some cases the inapplicability, of the approaches in certain domains. In [13, 20, 10, 6, 1] it is mentioned that characterizing workloads of real-time systems using *a priori* worst-case

execution times can lead to poor resource utilization, particularly when the difference between WCET and actual execution time is large. It is stated in [17, 1] that accurately measuring WCET is often difficult and sometimes impossible. In response to such difficulties, techniques for detection and handling of deadline violations have been developed [8, 18, 17]. Paradigms which generalize the execution time model have also been developed. Execution time is modeled as a *set* of discrete values in [9], as an interval in [19], and as a probability distribution in [10, 20, 2]. Most models consider execution time to apply to the job atomically; however, some paradigms [12, 18] view jobs as consisting of mandatory and optional portions; the mandatory portion has an *a priori* known execution time in [12], and the optional portion has an *a priori* known execution time in [18]. Most of these approaches assume that the execution characteristics (set, interval, or distribution) are known *a priori*. Others have taken a hybrid approach; for example, in [6] *a priori* worst case execution times are used to perform scheduling, and a hardware monitor is used to measure *a posteriori* task execution times for achieving adaptive behavior. The approach most similar to the one presented in this paper is described in [4], where resource requirements are observed *a po steriori*, allowing applications which have not been characterized *a priori* to be accommodated. Also, for those applications with *a priori* characterizations, the observations are used to refine the *a priori* estimates. These characterizations are then used to drive resource availability based algorithmic and period variation within the applications.

To further address these issues, we consider the problem of resource management in real-time systems with unpredictable, dynamically changing workloads. In Section 2, we present a load balancing example. Section 3 illustrates how this load balancing problem can be mapped to a control theoretic framework and identifies some of the problems and concerns of applying control theory to problems in resource management.

2 Load Balancing

In order to perform resource management for large, distributed real-time systems that have execution times and resource utilizations which cannot be characterized *a priori*, it is important to reallocate resources according to continuous feedback from the system. To illustrate this point, we consider an objective of load balancing for fixed workloads.

We will now consider a mathematical model for load balancing, and some associated heuristics. Consider the following function definitions:

$U(P_i, h_j)$ - the utilization of process P_i on host h_j

$U(h_j) = \sum_{P_i \in h_j} U(P_i, h_j)$ - the total utilization of h_j

The goal of load balancing is to keep the load variation across all the hosts in the system below some threshold ε. We state this goal as:

$$\max_j(U(h_j)) - \min_j(U(h_j)) \quad \leq \quad \varepsilon \qquad (1)$$

The observed difference between the host with the highest load and the host with the lowest load is the regulated variable in the system that we monitor and affect via reallocation actions. The feedback control policy is defined as follows:

1) Host load information is passed to the Load Balancer. The Load Balancer determines the state of the system by computing the difference between the highest and lowest observed loads.

2) The Load Balancer compares this value to the value of the threshold ε.

3) A reallocation is made by the Load Balancer.

Based on this architecture, we seek to utilize control theoretic methodologies to develop load balancing algorithms that effectively balance the load in systems with dynamically changing, unpredictable workloads.

To gain some insight into this problem, we now present an example of load balancing according to the architecture described above using a heuristic reallocation policy. For simplicity, we assume that a new process is assigned to the host with the lowest load, and a reallocation moves the process with the lowest CPU utilization from the host with the highest load to the host with the lowest load. In our examples, we set the threshold $\varepsilon = 0.10$. Figure 3 shows an initial mapping of processes onto hosts of a distributed system.

In Figure 4, we see the values of the function $U(h_j)$ for each host. Based on our definition of load balancing, the load across all the hosts is balanced. In Figure 5, new applications are placed on h_1 and h_3. As a result, the load across all the hosts becomes unbalanced. This fact is reflected in Figure 6.

The host monitors detect the unbalanced load, and a reallocation takes place. Based on our heuristics, the process with the lightest load, namely EDM, is moved from h_3 to h_4, since h_4 is the host with the smallest load. This reallocation is shown graphically in Figure 7. In Figure 8, we see the resulting values for $U(h_j)$. In this case, by applying a heuristic, our goal of load balancing was achieved.

3 Feedback Control Formulation

In order to apply control theory to the load balancing problem described in the preceding section, we first need to develop a feedback control system architecture as dictated by Figure 1. Figure 2 depicts a standard feedback block diagram for control law design. In the context of load balancing, the block labeled 'Plant' constitutes the process to be controlled which corresponds to a collection of host processors with associated communication links, the actuators which implement the reallocation of processes, and the sensors which monitor CPU load on each host. The Plant is modeled as having two types of input: exogenous inputs, which correspond to new processes that are spawned in a dynamic real-time environment, and control inputs which consist of commands to allocate new processes and/or reallocate existing ons. The Plant is modeled as having two types of output: a performance output, which in this case is given by the performance measure (1), and a measured output which corresponds to the CPU utilization of each host. The block labeled 'Controller' analyzes the host CPU utilization information (measured output) and, according to a prescribed policy, issues appropriate allocation and/or reallocation commands (control inputs). Thus the Controller corresponds to the Load Balancer block in Figure 1.

Within this framework, the following key issues arise as we endeavor to apply control theory to the problem of developing suitable reallocation policies for load balancing. First, performance requirements need to be precisely formulated. For the load balancing example, the objective is to achieve the performance bound (1) for a specified value of ε. Typically, the control system designer is interested in internal stability in addition to performance metrics associated with regulating the performance output. This motivates the need for precise stability definitions for dynamic real-time computing systems and associated analysis tools. Clearly, formal characterizations of stability must be consistent with phenomena observed in practice such as thrashing [5]. In the context of load balancing, it may happen that the performance bound (1) is satisfied yet processes are constantly being reallocated resulting in comparable though unnecessarily high host loads due to the increased overhead that is incurred. Ad-hoc techniques such as imposing transfer limits have been used in practice to reduce these unwanted effects. Other important performance characteristics such as robustness to model uncertainty (e.g. uncertain execution times) and fault tolerance (e.g., host or communication link failure) must also be translated into precise control-theoretic terms.

Also of paramount importance are appropriate mathematical models for the Plant (process to be controlled, sensors, and actuators) that accurately predict the complex input-output and internal behavior. The form taken by such models will dictate the types of control techniques that are applicable. In particular, model complexity issues such as system classification (e.g., continuous-variable vs. discrete-event vs. hybrid), dynamic order, and uncertainty structure will impact controller complexity. The discrete-event nature of dynamic real-time systems is evident from the load balancing example of the preceding section. Moreover, the strong tie between real-time computing systems and flexible manufacturing systems suggests that existing techniques for modeling, simulation, and control of the latter class might be applicable to the former class. On the other hand, direct application of classical control methods (e.g., PID control) has produced positive results in some applications. [15, 16]. Thus, assessing the trade-off between model fidelity and complexity is of fundamental importance. Engineering judgment suggests starting with the simplest models first and increasing complexity as increased fidelity becomes necessary.

4 Concluding Remarks

Resource management in dynamic systems with real-time constraints is a difficult problem in general. Our preliminary objective is to cast the problem of resource management for dynamic real-time systems in control-theoretic terms. Then, by applying control theory to resource management problems, we will proceed to develop better load balancing and QoS balancing techniques and to enhance the stability of dynamic real-time systems.

5 References

[1] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 3-13, IEEE Computer Society Press, 1998.

[2] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, 123-132, IEEE Computer Society Press, 1998.

[3] T.P. Baker, "Stack-based scheduling of realtime processes," *Journal of Real-time Systems*, **3(1)**, March 1991, 67-99.

[4] S. Brandt, G. Nutt, T. Berk and J. Mankovich, "A dynamic quality of service middleware agent for mediating application resource usage," in *Proceedings of the 19th IEEE Real-Time Sys. Symposium*, 307-317, IEEE Computer Society Press, 1998.

[5] D.L. Eager, E.D. Lazowska and J. Zahorjan, "Adaptive Load Sharing in Homogeneous

Distributed Systems," IEEE Trans. on Software Eng., vol. SE-12, no. 5, May 1986, pp.662-675.

[6] D. Haban and K.G. Shin, "Applications of real-time monitoring for scheduling tasks with random execution times ," *IEEE Transactions on Software Engineering*, **16(12)**, December 1990, 1374-1389.

[7] Robert D. Harrison Jr., "Combat system prerequisites on supercomputer performance analysis," in *Proceedings of the NATO Advanced Study Institute on Real Time Computing*, NATO ASI Series **F(127)**, 512-513, Springer-Verlag 1994.

[8] F. Jahanian, "Run-time monitoring of real-time systems," in *Advances in Real-time Systems*, Prentice-Hall, 1995, 435-460, edited by S.H. Son.

[9] T.E. Kuo and A. K. Mok, "Incremental reconfiguration and load adjustment in adaptive real-time systems," *IEEE Transactions on Computers*, **46(12)**, December 1997, 1313-1324.

[10] J. Lehoczky, "Real-time queueing theory," in *Proceedings of the 17th IEEE Real-Time Systems Symposium*, 186-195, IEEE Computer Society Press, 1996.

[11] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, **20**, 1973, 46-61.

[12] J.W.S. Liu, K.J. Lin, W.K. Shih, A.C. Yu, J.Y. Chung and W. Zhao, "Algorithms for scheduling imprecise computations," *IEEE Computer*, **24(5)**, May 1991, 129-139.

[13] K. Ramamritham, J.A. Stankovic and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Transactions on Computers*, **38(8)**, August 1989, 110-123.

[14] L. Sha, M. H. Klein, and J.B. Goodenough, "Rate monotonic analysis for real-time systems," in *Scheduling and Resource Management*, Kluwer, 1991, 129-156, edited by A. M. van Tilborg and G. M. Koob.

[15] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao, "The Case for Feedback Control Real-Time Scheduling," *EuroMicro Conference on Real-Time Systems*, June 1999.

[16] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-Driven Proportion Allocator for Real-Rate Scheduling," Operating Systems Design and Implementation (OSDI), February 1999.

[17] D.B. Stewart and P.K. Khosla, "Mechanisms for detecting and handling timing errors," *Communications of the ACM* , **40(1)**, January 1997,87-93.

[18] H. Streich and M. Gergeleit, "On the design of a dynamic distributed real-time environment," in *Proceedings of the 5th International Workshop on Parallel and Distributed Real-Time Systems*, 251-256, IEEE Computer Society Press, 1997.

[19] J. Sun and J.W.S. Liu, "Bounding completion times of jobs with arbitrary release times and variable execution times," in *Proceedings of the 17th IEEE Real-Time Systems Symposium*, 2-11, IEEE Computer Society Press, 1996.

[20] T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.C. Wu and J.W.S. Liu, "Probabilistic performance guarantee for real-time tasks with varying computation times," in *Proceedings of the 1st IEEE Real-Time Technology and Applications Symposium*, 164-173, IEEE Computer Society Press, 1995.

[21] J. Verhoosel, L. R. Welch, D. K. Hammer, and E. J. Luit, "Incorporating temporal considerations during assignment and pre-run-time scheduling of objects and processes," *Journal of Parallel and Distributed Computing*, ***36(1)***, July 1996, 13-31, Academic Press.

[22] L. R. Welch, A. D. Stoyenko, and T. J. Marlowe, "Modeling resource contention among distributed periodic processes specified in CaRT-Spec," *Control Engineering Practice*, **3(5)**, May 1995, 651-664.

[23] J. Xu and D.L. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations," *IEEE Transactions on Software Engineering*, **16(3)**, March 1990, 360-369.

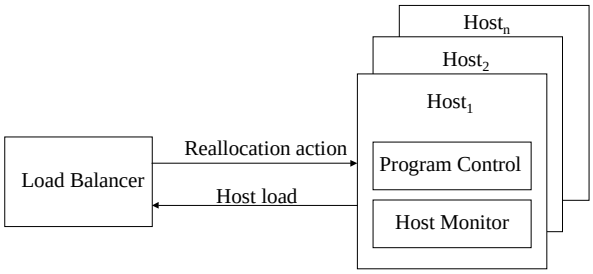

Figure 1. Load Balancing Architecture

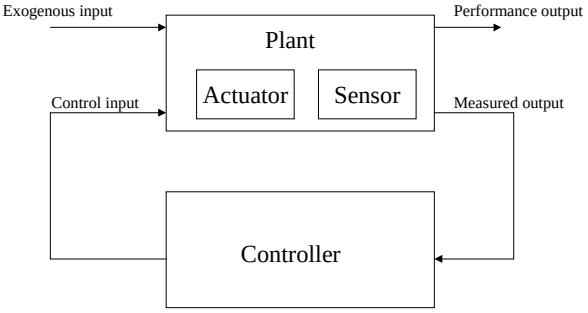

Figure 2. Feedback Control System Architecture

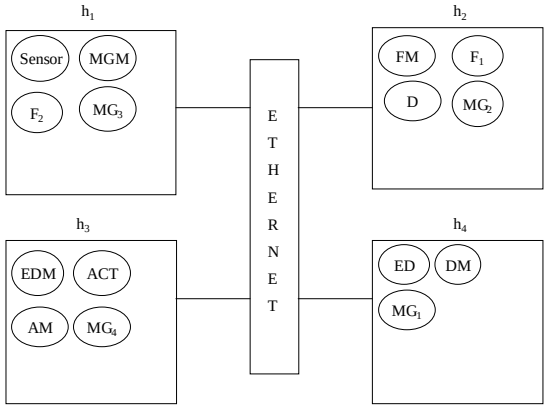

Figure 3. Initial Process-Host Mapping

Pi	h1	h2	h3	h4
Sensor	0.25			
FM		0.23		
F1		0.1		
F2	0.07			
EDM			0.11	
ED				0.17
AM			0.22	
ACT			0.14	
DM				0.29
D		0.2		
MGM	0.15			
MG1				0.18
MG2		0.16		
MG3	0.13			
MG4			0.14	
U(hj)	0.6	0.69	0.61	0.64

Figure 4. Balanced Load

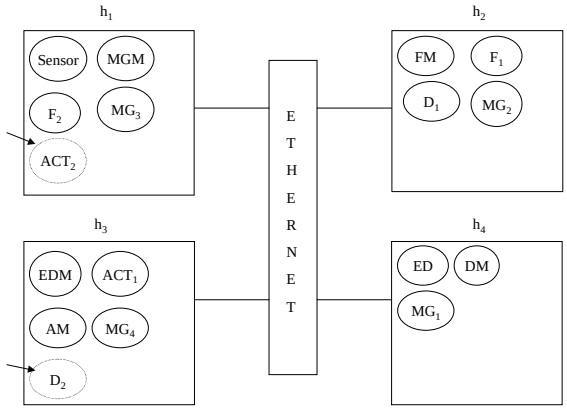

Figure 5. Load Increased

Pi	h1	h2	h3	h4
Sensor	0.25			
FM		0.23		
F1		0.1		
F2	0.07			
EDM			0.11	
ED				0.17
AM			0.22	
ACT1			0.14	
ACT2	0.12			
DM				0.29
D1		0.2		
D2			0.23	
MGM	0.15			
MG1				0.18
MG2		0.16		
MG3	0.13			
MG4			0.14	
U(hj)	0.72	0.69	0.84	0.64

Figure 6. Unbalanced Load

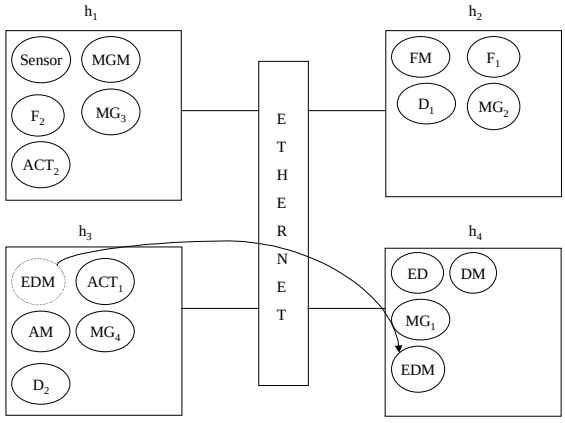

Figure 7. New Process-Host Mapping

Pi	h1	h2	h3	h4
Sensor	0.25			
FM		0.23		
F1		0.1		
F2	0.07			
EDM				0.11
ED				0.17
AM			0.22	
ACT1			0.14	
ACT2	0.12			
DM				0.29
D1		0.2		
D2			0.23	
MGM	0.15			
MG1				0.18
MG2		0.16		
MG3	0.13			
MG4			0.14	
U(hj)	0.72	0.69	0.73	0.75

Figure 8. After Reallocation