Quality of Service Management in Distributed Asynchronous Real-Time Systems

Binoy Ravindran

The Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061, USA
binoy@vt.edu

Abstract. This paper presents adaptive resource management techniques that achieve the timeliness quality of service (QoS) requirements of distributed real-time systems that are "asynchronous" – both in the sense that processing and communication latencies do not necessarily have known upper bounds, and in the sense that event arrivals are non-deterministically distributed. Examples of such systems include the emerging generation of computer-based, command and control systems of the U.S. Navy. To enable the engineering of such systems, we present resource management middleware strategies that enforce the timeliness QoS requirements of the system. The middleware performs QoS monitoring and failure detection, QoS diagnosis, and reallocation of resources to adapt the system to achieve acceptable levels of QoS. Experimental characterizations of the middleware using a distributed asynchronous real-time benchmark illustrate its effectiveness for adapting the system for achieving the desired QoS during overloaded situations.

1 Introduction

Real-time, military, computer-based command and control (C2) systems such as the surface combatants of the U.S. Navy are characterized by load patterns that do not have known upper bounds [3]. This is primarily due to the difficulty in estimating upper bounds on the size of data and event streams that they must process. Size of the data stream refers to the number of sensor reports that such systems must periodically process for decision making and reaction. Size of the event stream refers to the arrival rate of events that trigger computations in the system that perform mission critical tasks such as reacting to a hostile threat by detonating weapons. We call such real-time systems "asynchronous" real-time systems, as the processing and communication latencies do not have known upper bounds (due to unknown upper bounds on data stream sizes) and event arrivals are non-deterministically distributed (due to unknown arrival patterns of events).

Classical real-time computing research focuses on "hard" real-time computing for synchronous (in the sense that processing and communication latencies have known upper bounds and event arrivals are deterministically distributed),

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 489–496, 1999.

device level, sampled data monitoring and regulatory control - usually centralized but occasionally distributed[1, 4, 5, 9]. These techniques cannot be practically employed or adapted for systems that are distributed and asynchronous. Distributed asynchronous real-time computer systems and their applications are inherently dynamic, and thus require end-to-end adaptive real-time resource management [2]. We argue that the conventional "hard"/"soft" dichotomy is too coarse, and timeliness is better treated as a quality of service (QoS) for activities in such systems. The paucity of concepts, theories, methodologies, and techniques that allow engineering of, and reasoning about, asynchronous distributed real-time computing systems–and the consequent absence of appropriate commercial real-time operating system and system software (e.g., middleware) products–forces system engineers to "home-brew" and construct ad hoc special purpose, and consequently expensive, technologies and solutions. A good example of this is the current generation of Navy surface combatants.

In response to the need for an infrastructure for engineering distributed asynchronous real-time systems, we present resource management strategies that can achieve their QoS requirements. The objective of resource management is to adapt the system at run-time to changing resource needs so that acceptable levels of timeliness QoS can be achieved. Adaptation is achieved by identifying and eliminating software bottlenecks and by discovering additional hardware resources. The resource management techniques are implemented as part of a middleware infrastructure for the emerging generation of shipboard computing systems using commercial-off-the-shelf (COTS) workstations. Further, the effectiveness of the techniques are evaluated using a distributed asynchronous real-time benchmark application that functionally approximates a shipboard C2 system.

The rest of the paper is organized as follows: Section 2 presents a generic model of real-time C2 systems. The generic model is derived after a study of Navy's Anti-Air Warfare (AAW) system [6] and is used to reason about asynchronous real-time systems and their load characteristics. Steps involved in the resource management process are described in Section 3. An experimental characterization of the resource management algorithms is presented in Section 4. Finally, the paper concludes with a summary of the current work and on-going efforts in Section 5.

2 A Generic Real-Time C2 System

Figure 1 shows a generic real-time C2 system. The control system consists of functions that perform *assessment* of the environment, *initiation* of actions, and monitoring and *guidance* of the actions to their successful completion. The interrelationship of the functions with the environment, and the intra-relationship of the functions among themselves are illustrated in Figure 1.

The assessment function repeatedly collects data from the environment through hardware sensing devices. The data is filtered, correlated, classified, and then used to determine the necessity of an action by the system. The assess-

ment function has a *continuous* (or cyclic) behavior, since the data collection and assessment is performed repeatedly throughout the operational life of the system. Typically, there is a timeliness objective associated with the completion of each execution cycle of the function, i.e., a bound on the time taken to review each of the elements of the data stream once. When an action is necessary, the assessment function generates an event that activates the initiation function.

The initiation function determines the action that needs to be taken and causes actuating devices to perform the action. Since the function executes in response to an event that can be triggered at any time, the initiation function has a *transient* behavior. The initiation function has, typically, a maximum latency requirement on its execution time, i.e., a latency requirement on the execution time of the function to process a single event. Upon initiation of the actions by the actuators, the monitoring and guidance function is notified.

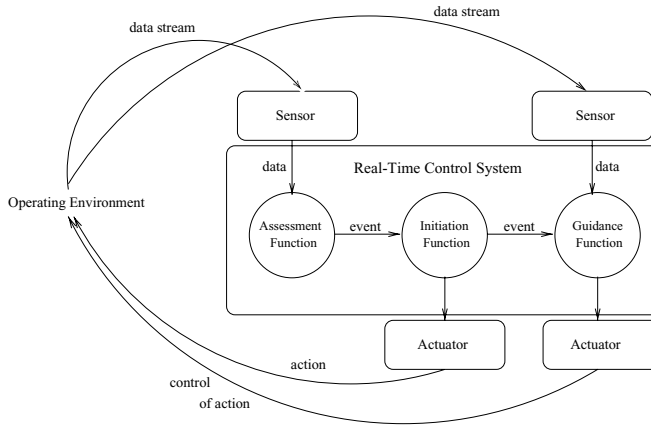


Fig. 1. A Generic Real-Time C2 System

The guidance function repeatedly uses sensor inputs to collect data, to monitor the actions that were initiated, and to guide the actuators to successful completion of the actions. Since the activation of the guidance function can begin and terminate at any time, the function is transient in behavior. Further, the function behaves like an assessment function once it is active, executing in a continuous manner. Hence, the guidance function has a *quasi-continuous* behavior. Observe that each transient activation of the function consists of a set of execution cycles. Guidance functions typically have two timeliness objectives: (1) completion time for one cycle and (2) deactivation time. Typically, it is more critical to perform the required processing before the activation deadline than it is to meet the completion time for each cycle.

After a careful study of Navy's AAW C2 system, we have observed that the load characteristics of control system functions are significantly influenced by: (1) size of the data stream (or the number of data items that the assessment

and guidance functions have to process during a single cycle), and (2) size of the event stream (or the arrival rate of events that trigger the execution of the initiation and guidance functions). For systems such as the AAW, data stream sizes (radar tracks) and event (threat) arrivals have neither known upper bounds nor deterministic distributions, respectively. Thus, we call such real-time systems, "asynchronous" real-time systems.

3 Adaptive Resource Management Steps

The steps involved in the resource management process are illustrated in Figure 2. The real-time control system components are monitored for conformance to specified QoS requirements. QoS violations of the system are detected and are reported to diagnosis functions. Diagnosis functions identify the causes of QoS violations. Further analysis identifies possible recovery actions to improve the QoS. Allocation analysis is performed to determine the optimal way to execute the selected actions. We describe the algorithms that are used in each of these steps in the subsections that follow.

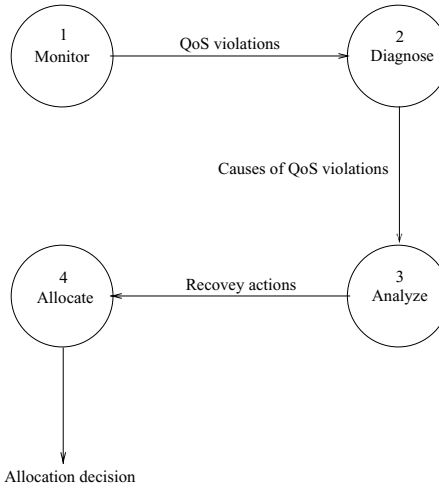


Fig. 2. Adaptive Resource Management Steps

3.1 QoS Monitoring

Monitoring of timeliness QoS involves collection of time-stamped events that are sent from application programs and synthesis of the events into function-level end-to-end QoS metrics. Analysis of a time series of the metrics enables detection of QoS violations. In this paper, we consider timeliness QoS management of the continuous assessment function.

We define the timeliness QoS of assessment functions in terms of a *minimum slack* value that must always be maintained on the cycle deadline of the function. A timeliness QoS violation of the assessment function is said to occur during an execution cycle when the observed cycle latency of the function exceeds the minimum slack requirement on the cycle deadline of the function. Further, such a violation has to be observed for a certain number of cycles that is larger than the maximum that is allowed in a "window" of recent cycles.

3.2 QoS Diagnosis

When a QoS violation occurs, diagnosis techniques are employed to determine: (1) the cause(s) of the violation, and (2) the possible action(s) that may improve the QoS and recover from the violation.

A timeliness QoS violation of the assessment function takes place due to the increase in execution latencies of some applications of the function, or due to the increase in communication latencies between some pairs of applications (that communicate) of the function, or both. Algorithms that perform timeliness QoS diagnosis identifies applications or communication links that are experiencing significant increase in latencies. In this paper, we consider a "local" QoS diagnosis technique. The technique is local in the sense that it considers only the applications of a control system function that has been reported for QoS violations. The diagnosis technique compares current latency of an application to its least latency in the same host assignment and at the same data stream size. The technique identifies the latency of an application to have increased if the end-to-end cycle latency of the function during the current cycle has exceeded the least function latency that was observed in the past by a significant amount.

3.3 Identifying Recovery Actions

Once the causes of a QoS violation are determined and a set of "unhealthy" applications are identified, further analysis is performed to determine possible recovery *actions* for the applications that will improve the timeliness QoS of the function. We consider an algorithm that analyzes the changes in the data stream size and the host load of an unhealthy application and determines the recovery action as follows:

- If the data stream size of the application has increased significantly and the load of the host of the application has not increased by a large factor, then the recovery action is to *replicate*. The replicas of the application can share the additional data items, process it concurrently, and thereby reduce the end-to-end function cycle latency.
- If the data stream size of the application has not increased significantly and the load of the host of the application has increased by a large factor, then the action is to *migrate*. The application may be residing on a host resource that is heavily "loaded" and may be subjected to increased contention. By migrating the application to a relatively less loaded resource, the resource

contention of the application can be reduced. This can reduce the function latency.

3.4 Allocation Analysis

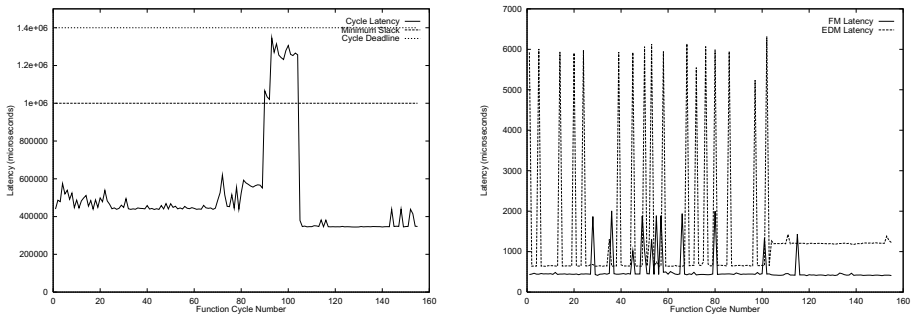
The QoS analysis algorithm discussed in the previous section produces a set of (*unhealthy-application*, *action*) pairs. To enact the recovery actions for the unhealthy applications, hardware resources must be allocated to the actions. We consider a resource allocation algorithm that uses a set of candidate (application,action) pairs as its input and selects the hardware resources–host and LAN–for performing the recovery actions.

The algorithm determines the "best" host to replicate or migrate an unhealthy application, and a LAN of the host for the inter-application communication needs of the application. The algorithm only considers the set of hosts where an application is prepared for execution and the set of LANs of each such hosts. The best host and a LAN of the host is determined using a "fitness" function that simultaneously considers host and LAN load information. Load information of hosts and LANs are characterized as trend values over a moving set of load index measurements by determining the slope of a simple linear regression line that plots the load index values as a function of time. The load index value of a host is determined as a function of CPU utilization, CPU ready-queue length, and free available memory, and that of a LAN is determined as a function of the communication activity detected at network interfaces of hosts in the LAN, respectively. The function definitions can be found in [7]. The best host of a candidate application is determined as the host that has the minimum fitness value among all the eligible hosts of the application.

4 Experimental Evaluation

We evaluate the effectiveness of the resource management algorithms through an experimental study. A distributed asynchronous real-time benchmark application is used in the experimental characterizations. The benchmark consists of an assessment function called Sensing that consists of a sensor (a simulator program called Sensor), a filter manager program (called FM), one or more replicas of a filter program (called Filter), an evaluate and decide manager (called EDM), and one or more replicas of an evaluate and decide program (called ED). Details of the benchmark can be found in [8]. The hardware test-bed of the experimental environment consists of a network of SUN Ultra workstations and Pentium-based PCs that are interconnected using 100MBPS Ethernet LANs.

The experiment described here illustrates a scenario where the middleware recovers the Sensing function of the benchmark from timeliness QoS violations during high data stream size situations. For the experiment, we consider an initial data stream size of 1000 data items, a cycle deadline of 140 milliseconds, and a minimum timeliness slack value of 100 milliseconds (i.e., approximately 72% of the deadline). The maximum number of allowable violations and the



(a) (b)

Fig. 3. Latencies of (a) Sensing Function and (b) FM and EDM Programs During High Data Stream Sizes

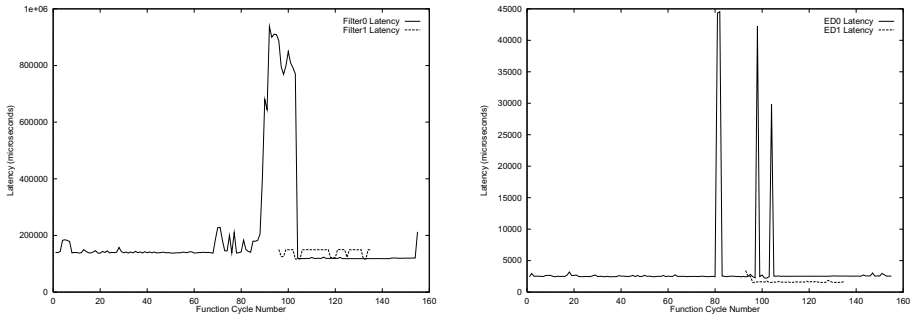
window size is defined as 15 and 20, respectively. The experiment is started by gradually increasing the data stream size from 1000 to 2000 data items. As seen in Figure 3(a), the cycle latency of the function increases and eventually exceeds the minimum slack requirement. Figures 3(b), 4(a), and 4(b) show the corresponding latencies of the FM, EDM, Filter, and ED programs of the function, respectively.

The middleware detects the timeliness QoS violation and performs QoS diagnosis. The result of the diagnosis is to replicate Filter and ED programs, as both the programs are experiencing high data stream sizes. The resource allocation algorithm is used to compute an allocation decision, i.e., the best hosts and LANs for the use of the replica programs. The allocation decision is enacted by the middleware. The newly created replica processes synchronize with other executing programs and start processing the data. Figures 4(a) and 4(b) illustrate the latencies of the replicas of Filter and ED programs, respectively. The replication of the programs and the sharing of the data stream among the replicas cause the function cycle latency to decrease and improves its timeliness QoS. This is observed in Figure 3(a).

5 Conclusions

This paper presents adaptive resource management techniques that achieve the QoS requirements of continuous computations in distributed asynchronous real-time systems. Experimental characterizations of the resource management strategies illustrate its effectiveness in recovering from timeliness QoS violations that occur during high data stream size situations. A benchmark that functionally approximate shipboard C2 systems is used in the experimental study.

The resource management middleware described in this paper has been incorporated into the experimental test-bed of the Navy. On going work includes development of resource management algorithms for transient and quasi-continuous computations that can achieve their desired timeliness QoS.



(a) (b)

Fig. 4. Latencies of (a) Filter and (b) ED Programs During High Data Stream Sizes

References

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

[2] R. Clark, E. D. Jensen, A. Kanevsky, et al. An adaptive, distributed airborne tracking system. In *Parallel and Distributed Processing*, volume 1586, pages 353–362. Springer-Verlag, April 1999. Lecture Note in Computer Science.

[3] R. D. Harrison Jr. Combat system prerequisites on supercomputer performance analysis. In *Proceedings of The NATO Advanced Study Institute on Real-Time Computing*, volume F127 of *NATO ASI*, pages 512–513, 1994.

[4] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123, August 1989.

[5] L. Sha, M. H. Klein, and J. B. Goodenough. Rate montonic analysis for real-time systems. In A. M. van Tilborg and G. M. Koob, editors, *Scheduling and Resource Management*, pages 129–156. 1991.

[6] L. R. Welch. Large-grain, dynamic control system architectures. In *Proceedings of The Workshop on Parallel and Distributed Real-Time Systems*, April 1997.

[7] L. R. Welch et al. Instrumentation, modeling and analysis of dynamic, distributed real-time systems. In *International Journal of Parallel and Distributed Systems and Networks*, 1999. Special Issue on Measurement of Program and System Performance, Accepted for publication. To appear.

[8] L. R. Welch and B. A. Shirazi. A dynamic real-time benchmark for assessment of qos and resource management technology. In *Proceedings of The Fifth IEEE Real-Time Technology and Applications Symposium*, June 1999.

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