A Hierarchical Control System for Dynamic Resource Management *

Kurt Rohloff, Jianming Ye, Joseph Loyall, Richard Schantz
BBN Technologies
10 Moulton St., Cambridge, MA, USA
{krohloff, jye, jloyall, schantz}@bbn.com

Abstract

We present a hierarchical control system for the dynamic resource management of a distributed real-time embedded (DRE) system. This DRE is inspired by the DARPA Adaptive and Reflective Middleware Systems (ARMS) program. The goal of the control system is to simultaneously manage multiple resources and QoS concerns. The control system is scalable and uses a utility-driven approach for decision-making and performance evaluation. The control system is designed to be easily adaptable to other multi-tiered DRE systems.

1 Introduction

Large distributed real-time embedded systems are often designed with static resource management strategies tailored for specific goals or missions. These rigid resource allocation strategies are incapable of adapting to changing system goals, resource levels and operating environments. This inability to adapt can cause DRE systems to fail to meet end-to-end quality of service (QoS) requirements when conditions change.

We present a hierarchical control system for the dynamic resource management of hierarchical DRE systems that is capable of simultaneously managing multiple resources and QoS concerns. Dynamic resource management has the capability to achieve much higher performance in a constrained resource system than static resource management approaches.

The DRE application area is inspired by the DARPA Adaptive and Reflective Middleware Systems (ARMS) program. The ARMS program is aimed at developing multi-level middleware services to manage computational and communication resources across a common Navy Total Ship Computing (TSC) environment which is composed of many distinct, distributed elements. This environment is currently being developed for the DD(X) family of shipboard surface ships by a team led by Raytheon and Lockheed Martin. The Navy program serves as a transition target for the ARMS technology development activities. The ARMS system can be decomposed into multiple missions and missions can be decomposed into multiple submissions called application strings or strings. This multi-tiered hierarchy is a common aspect of many DRE systems.

A key element of the control system we are designing for these DRE systems is a utility-driven approach for decision-making and performance evaluation with respect to resource allocation in the controllers. Utility is computed for each element in the system hierarchy (string, mission, system) and is a measure of that element's ability to perform its desired tasks. The allocation of system resources is dynamically managed to locally maximize utility at each level of the system hierarchy with individual controllers deployed for the whole system and all missions and its strings.

The general philosophy for the control system is a bottom-up approach to dynamic resource management. At the lowest levels, controllers perform fast, frequent, local tunings of system behavior, while at the highest levels, controllers perform less frequent, but more aggressive control actions. We feel that this bottom-up scalable control approach can be easily applied to other hierarchical DRE systems. Other dynamic control approaches to meet QoS requirements are in [1, 2].

The next section describes key elements of the multi-tiered hierarchical system. The controllers and utility measures for dynamic resource management are outlined in Section 3. We conclude the paper and discuss several avenues of future work in Section 4.

2 System Architecture

Properties of DRE systems can be understood via aspects of both their resources and applications running on those resources. The resource aspects of DRE systems include the

*This work has been supported by DARPA under contract number NBCHC030119. The authors would like to acknowledge the contributions of our ARMS teammates and other researchers. Approved for Public Release, Distribution Unlimited

computation and communication resources of the system. The computational resources are a set of general purpose computer hosts. The communication resources in the system are the communication links formed between various hosts in the systems and the attributes of these links such as bandwidth, maximum delay and operating modes.

Hosts are assumed to be grouped into pools or clusters of computing resources based on their physical locations. Pools are managed independently of one another by local pool managers. Communication between hosts in the same pool is assumed to be generally inexpensive, while hosts in a pool share limited communication gateways to hosts in other pools. Therefore, communications between hosts is partitioned into intra- and inter-pool communications.

Software applications are deployed onto the computational resources and can be viewed at multiple levels of abstraction. At the lowest level of abstraction, applications run on hosts and perform work requiring certain computing resources.

At the next highest level of abstraction, an application string, or string, is a logical sequence of applications that sequentially process information with unique starting and ending applications. Strings are generally deployed across multiple hosts and pools, so string controllers need to manage inter- and intra-pool communication. Strings generally perform work subject to end-to-end QoS requirements. Two or more strings may share an application.

At the penultimate level of abstraction, a mission is a group of strings that cooperate to achieve common goals. At the highest level of abstraction, the system incorporates all running missions and resources those missions have access to. A schematic of the system-mission-string decomposition can be seen in Figure 1.

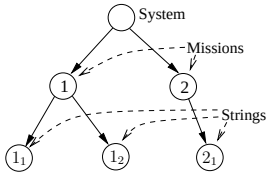

Figure 1. System-mission-string hierarchy.

The ARMS system has software components called the Infrastructure Allocator (IA) that allocates applications to hosts and a Bandwidth Broker (BB) that allocates bandwidth on intra- and inter-pool communication links. The IA actuates the control actions of the control system by (re)allocating computational resources, and the BB actuates the control actions of the control system by (re)allocating communication resources. It is assumed that the DRE systems considered in this paper have similar software components to actuate control actions. It is also assumed that system status measurement information is shared by a distributed publish-and-subscribe service called RSS (Resource Status Service). RSS allows system components to improve efficiency by not having to redundantly gather and distribute status information independently([3]).

3 Control System Design

The hierarchical control system uses a set of utility functions to evaluate the performance of strings and missions in the system against current resource allocations. The control system also uses the utility estimation function to estimate the desirability of various control actions with respect to the future performance and utility of the system. The control system chooses control actions that would result in a higher level of estimated utility.

If the system has enough unused system resources, the system could allocate resources to previously undeployed missions or application strings to boost its overall utility and performance. Conversely, if resource contention were to occur due to an over-deployment of missions (possibly due to resource failure among other possible causes), then the performance and utility of the deployed missions would drop. A drop in the measured utility indicates to the controllers that the allocation of resources should be adjusted in an attempt to relieve the resource contention and raise the measured utility.

An overview of the utility functions is in Subsection 3.1. The design of the control system used to maintain a high level of measured utility in multi-tiered DRE systems is outlined in Subsection 3.2.

3.1 Utility Function

We use a set of hierarchical utility functions to measure the performance of the system that follows the system-mission-string hierarchy outlined in the introduction. Utility functions are defined for the system, each of the missions and each of the application strings that measure the performance of these entities under current resource allocation. More formally, at a given time t:

- $U(t)$ is the utility of system performance.
- $U_i^m(t)$ is the utility of mission i.
- $U_i^{s_j}(t)$ is the utility of string j of mission i.

We define the system-level utility, $U(t)$ to be a weighted sum of the mission-level utilities:

$$U(t) = \Sigma_{i=0}^{M} w_i^m U_i^m(t).$$

The weight factor w_i^m is a measure of the relative importance of mission i .

Similarly, the mission's ability to complete its required tasks depends on the ability of its strings to complete their desired tasks, so mission utility $U_i^m(t)$ is a weighted sum of the mission's string-level utilities:

$$U_i^m(t) = \Sigma_{j=0}^{S_i} w_i^{s_j} U_i^{s_j}(t)$$

The weight factor $w_i^{s_j}$ is a measure of the relative importance of string s_j of mission i.

The utility of string s_j from mission i depends on the timeliness, quality, and throughput of information processed by the string. These factors are an indication of how well the string can process and transmit information. Timeliness (denoted $T_i^{s_j}(t)$) is a measure of the application string's ability to meet end-to-end realtime requirements. Quality (denoted $q_i^{s_j}(t)$) is a measure of how useful the information processed by an application string is. Throughput (denoted $Th_i^{s_j}(t)$) is the rate at which information to be processed is sent to the string. The mapping of the factors $T_i^{s_j}(t), q_i^{s_j}(t), Th_i^{s_j}(t)$ to the utility of the string may vary from string to string, so we define $U_i^{s_j}(t)$ to be computed by a generic function $F_i^{s_j}(\cdot,\cdot,\cdot)$:

$$U_i^{s_j}(t) = F_i^{s_j}(T_i^{s_j}(t), q_i^{s_j}(t), Th_i^{s_j}(t)).$$

The utility of a string $U_i^{s_j}(t)$ is periodically computed by its string controller and published on RSS so that the higher level controllers can compute $U_i^m(t)$ and $U(t)$. Future research will focus on obtaining appropriate expressions for timeliness, quality, and throughput.

3.2 Control System

In order to hierarchically control the system via dynamically allocated system resources to maximize system utility, controllers are deployed with one controller for every string and mission and one system controller. A diagram of the control system in conjunction with a DRE system with the system-mission-string hierarchy can be seen in Figure 2. At the top of the diagram, the high-level DRE system is controlled by the system controller. At the middle level, two missions are controlled by local mission controllers. At the lowest level, one mission has two strings and the other mission has one string. The strings all have local string controllers. All of the controllers in the hierarchy communicate with their parents and/or children to facilitate communication between control layers in the bottom-up control design.

At initialization, the IA performs a system-wide allocation of resources. Therefore, immediately after initialization, all of the controllers in the system would be able to take a baseline utility measurement to ideally be maintained by the controllers. This baseline measurement is stored in local memory for future reference. If any controller observes a significant drop in its measured utility below the measured baseline, this signals the controller that it should take actions to attempt to raise its measured utility.

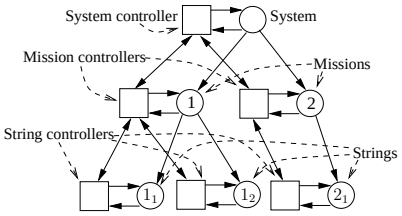

Figure 2. The control hierarchy.

Under normal operating conditions, (when there is no resource contention or resource failure) a string's measured utility should fluctuate minimally around its stored baseline. It remains an area of future research to determine how the controllers should determine when a measured drop in utility is just a normal fluctuation and when a drop is a symptom of resource contention or resource failure. Another open area for research is to determine how often the controllers should measure their local utility in order to ensure a sufficiently fast control response to resource contention or resource failure.

The control system uses a hierarchical control philosophy that is fundamentally bottom-up. The low level controllers are generally fast and responsive, while the high level controllers have the ability to take more aggressive control actions. Because higher level control actions are more invasive, the higher level controllers take more time to better estimate which of their control actions will maximize their local utility. Because local controllers in this design attempt to greedily maintain their local utility, the bottom-up control philosophy limits local, fast utility gains that are potentially detrimental to the overall system utility.

String controllers perform fast low-level tuning of quality and throughput in order to maintain their local string utility. If a string controller is unable to maintain its local utility, its mission controller performs limited resource redeployments to benefit local strings. If a mission controller is unable to maintain its local utility, the mission controller sends a request to the system controller to reinitialize system resources. The system controller has the ability to request that the IA perform a full reinitialization of system resources if the controllers are unable to take any action that would sufficiently raise the measured system utility. The controllers interact with each other through direct communications, but the controllers receive information about system resources or performance through RSS.

A drop in string utility could be caused by either resource contention or failure, but on resource failures, the string controller is expected to receive notification about the failures from RSS. In the absence of a notification from RSS

indicating otherwise, the string controller assumes drops in utility are caused by resource contention.

Generally the quality ($q(t)$) and the throughput ($Th(t)$) of an application string can be directly controlled by the string's controller by adjusting applications in a string, but the timeliness ($T(t)$) cannot. However, the timeliness of a string can be influenced by tuning the quality and throughput of information processed by a string. When a string controller observes that $U_i^{s_j}(t)$ is significantly below its measured baseline, the string controller attempts to decrease the string's quality and throughput. Any observed improvement in timeliness by decreasing quality and throughput will not be instantaneous, so incremental decreases are made in both quality and throughput on the utility measurement cycles. Quality and throughput are continually decremented until a local maximum of the measured string utility $U_i^{s_j}(t)$ is found. If the local maximum is not sufficiently close to the utility baseline, the string controller sends a signal to the mission controller that the mission controller should attempt to relieve the string's observed resource contention. It remains an open problem to determine how aggressively the string controllers should decrement quality and throughput in attempts to maintain their local utility.

When a mission controller receives a signal that one of its strings is unable to maintain its utility, the mission controller attempts to redeploy portions of the string. As indicated in Section 2, the system resources are arranged in pools and the strings are possibly deployed across multiple pools. The inter-pool communication links generally cause the greatest contention, so the mission controller attempts to redeploy portions of its strings from one pool to another pool to avoid problematic communication links.

To perform this control action, the mission controller first queries the BB via RSS to find out which of the communication links along the path of the string have the highest contention. The mission controller then queries the IA to find what other pools in the system would be able to accept the original substring(s). After receiving the list of candidate pools to move the substring(s) to, the mission controller queries the BB via RSS for timeliness estimates of the string for each of the new pool substring assignments. The mission controller then moves the substring to the pool that would result in the highest string utility due to the given timeliness estimates.

Resource failures that result in drops in a string's utility are handled by the mission controller. All strings that use a failed resource will generally have zero utility after the failure. Additionally, the string controller would be unable to adjust its quality and throughput to regain lost utility, so it would fall on the mission controller to attempt to redeploy the portions of all of its strings using the failed resources. This redeployment action would be performed in the same manner that the mission controller responds to insufficient string utility only now there may be multiple strings involved.

If the partial redeployment actions of the mission controller do not raise its strings' utility to its desired threshold, the mission controller sends a signal to the system controller requesting a system-wide reallocation of resources.

Upon receiving a request from a mission controller requesting a system-wide reallocation, the system controller sends a request to the IA for a system-wide reinitialization of resources. Due to the bottom-up control philosophy, the relatively drastic reinitialization control action is only taken if the lower level controllers are unable to sufficiently maintain their local utilities.

4 Conclusions and Future Work

We have presented a utility driven hierarchical controller design for multi-tiered DRE systems to dynamically manage system resources. Although only three levels of abstraction are considered here for the DRE and control systems, our design is easily scaled to any number of control levels where low level controllers take fast, limited actions and high level controllers take slow, more invasive actions.

It remains an area of future research to determine how the controllers should determine when a measured drop in utility is just a normal fluctuation and when a drop is a symptom of resource contention or resource failure. It also remains an open problem to determine how often the controllers should sample their local utility measurements and how aggressive the string controllers should be in their local tunings of string behavior.

Experiments need to be implemented and run to evaluate the performance of the resource allocation strategies. The control system will also be implemented in software for the DARPA ARMS program. The control system design will be refined to increase performance and reliability. The control system will also be implemented in a software prototype for the DARPA ARMS program, against the expected system configurations for the evolving DD(X) family of shipboard Navy systems.

References

[1] C. Lu, J. Stankovic, S. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems*, 23(1–2):85–126, July 2002.

[2] H. Wu, B. Ravindran, E. Jensen, and P. Li. Time/utility function decomposition techniques for utility accrual scheduling algorithms in real-time distributed systems. *IEEE Transactions on Computers*, 54(9):1138–1153, 2005.

[3] J. Zinky, J. Loyall, and R. Schapiro. Runtime performance modeling and measurement of adaptive distributed object applications. In *Proceeding of International Symposium on Distributed Object and Applications (DOA)*, Irvine, CA, 2002.