# Empirical Differences Between COTS Middleware Scheduling Strategies*

Christopher D. Gill, Fred Kuhns, and Ron K. Cytron
{cdgill, fredk, cytron}@cs.wustl.edu
Department of Computer Science
Washington University, St.Louis

Douglas C. Schmidt
schmidt@uci.edu
Electrical & Computer Engineering
University of California, Irvine

This paper has been submitted to the 4th International Symposium on Distributed Objects and Applications, Irvine, CA, October-November, 2002.

## Abstract

*The proportion of complex distributed real-time embedded (DRE) systems made up of commercial-off-the-shelf (COTS) hardware and software is increasing significantly in response to the difficulty and expense of building DRE systems entirely from scratch. In previous work, we showed how applying different scheduling strategies in middleware can allow COTS-based solutions to provide both assurance and optimization of real-time constraints for important classes of mission-critical DRE systems. There are few empirical studies, however, that help developers of COTS-based DRE systems to make crucial distinctions between strategies that appear similar in policy, but whose run-time effects may differ in practice.*

*This paper provides two contributions to the study of real-time quality of service (QoS) assurance and performance in COTS-based DRE systems. First, we examine in detail two hybrid static/dynamic scheduling strategies that should behave similarly according to policy alone, but that in fact produce different results under the same conditions, both in utilization and in meeting real-time assurances. Second, we offer recommendations based on these results for developers of mission-critical DRE systems, such as the Boeing Bold Stroke platform used in the Adaptive Software Flight Demonstration (ASFD) program under which our experiments were conducted. These contributions address and highlight the importance of the following issues to real-time scheduling in COTS environments: (1) careful mapping of scheduling policies into implementation mechanisms and (2) benchmarking and analysis of actual systems in representative operational environments.*

**Keywords:** Dynamic Scheduling Algorithms and Analysis, Real-Time Assurance and Optimization, Distributed Real-time and Embedded Systems, Mission Critical Systems, Quality of Service Issues, Middleware and APIs.

# 1 Introduction

## 1.1 Motivation: Distributed Real-Time and Embedded Systems

Distributed, real-time, and embedded (DRE) systems are becoming increasingly widespread and important. Examples of DRE systems include *autonomous agent teams*, *e.g.*, multi-robot environment mapping, *manufacturing process automation*, *e.g.*, high-performance assembly lines, and *defense systems*, *e.g.*, avionics mission computing systems. Although there are many types of DRE systems, they must all achieve the following capabilities:

- Managing connections and data transfer between distinct endsystems
- Offering predictable and efficient control over end-to-end system resources and
- Operating within computing and memory resource limitations imposed by stringent weight, cost, and power constraints.

Designing DRE systems that can offer strong assurances of real-time predictability, and yet are parsimonious in their use of limited computing resources is hard; building them on time and within budget is even harder. Therefore, studying real-time policies and mechanisms within commercial-off-the-shelf (COTS) systems empirically is essential to ensure mission-critical DRE systems can be built and maintained in a cost-effective manner.

Our previous work [1, 2] has quantified the benefits of applying multiple scheduling paradigms in COTS middleware to support the quality of service (QoS) demands of mission-critical DRE systems that possess a mix of hard and soft real-time requirements, such as avionics mission computing systems [3], mission-critical distributed audio/video processing [4, 5], and real-time robotic systems [6]. However, our previous work also showed that when more than one strategy is plausible for a given system based on policy alone, the best choice of which strategy to apply may not be obvious.[1]

*This work was supported in part by Boeing, DARPA ITO, DARPA contract F33615-00-C-1697 (PCES) and AFRL contracts F3615-97-D-1155/DO (WSOA) and F33645-97-D-1155 (ASTD/ASFD).

[1] We distinguish the *policy* of a scheduling strategy, *i.e.*, the *algorithm* for ordering operations it schedules, from the *mechanisms* used to implement that policy, *e.g.*, prioritized threads, queues, and timers.

This paper therefore presents additional empirical results and analysis of multi-paradigm COTS middleware to offer insights into a canonical choice between two hybrid static/dynamic scheduling strategies. We suggest guidelines for choosing between the strategies examined in this research, and posit a model and hypotheses to test that model for further investigation of these issues as future work.

The research presented in this paper was conducted in the context of a real-world mission-critical DRE application: a research operational flight program (OFP) designed for technology transfer to production avionics mission computing systems. This paper can therefore be viewed as a case study of the application of scheduling strategies in middleware for next-generation DRE systems. The results it presents apply to a class of DRE systems that cross-cuts application domains that manage both critical and non-critical real-time requirements.

## 1.2 Context: Real-Time CORBA Middleware

DRE systems have historically been custom developed in an *ad hoc* and inflexible manner. While many operational systems have been built this way, this development process failed to address the following challenges adequately: (1) reducing total ownership costs, (2) providing portable QoS management, and (3) tailoring resource provisioning to assure *critical* system requirements are met in the worst case, while recovering resources appropriately to improve *non-critical* performance in the average case. In recent years, the following technologies have converged to address these challenges:

**Distributed object computing (DOC) middleware.** DOC middleware is systems software that resides between the applications and the underlying operating systems, network protocol stacks, and hardware [7]. It offers clients portable language-independent and location-transparent invocation of methods on target object implementations [8]. DOC middleware simplifies application development by off-loading the tedious and error-prone aspects of distributed computing from application developers to middleware developers.

**Real-Time CORBA.** Real-time CORBA [9] is a DOC middleware standard that adds capabilities that support end-to-end predictability for remote operations to the original CORBA specification. It improves system predictability and bounds or avoids priority inversions, by supporting end-to-end management of system resources. To implement Real-time CORBA effectively, an Object Request Broker (ORB) must provide run-time support to automate many DRE features, such as connection management, marshaling/demarshaling, demultiplexing, language and OS independence, resource scheduling and load balancing, error handling and fault-tolerance, and security. However, first-generation ORBs did not provide features or optimizations to support DRE systems with stringent QoS requirements.

**The ACE ORB (TAO).** To meet the stringent QoS requirements of DRE systems, researchers at Washington University in St. Louis and the University of California, Irvine have developed a second-generation ORB called TAO [10], which is an open-source implementation of Real-time CORBA that supports efficient, predictable, and flexible DRE computing. Prior work on TAO has explored many dimensions of high-performance and real-time ORB design and performance, including scalable event processing [11], request demultiplexing [12], I/O subsystem [13] and protocol [14] integration, connection architectures [15], asynchronous [16] and synchronous [17] concurrent request processing, adaptive load balancing [18], meta-programming mechanisms [19], and IDL stub/skeleton optimizations [20].

**Kokyu Multi-paradigm Scheduling Framework.** To increase responsiveness to varying operational environments, we have recently [2] extended our prior research on static [10] and dynamic [3] scheduling for Real-time CORBA by incorporating a *strategized scheduling framework* called *Kokyu*[2] within the TAO Real-Time Event Service. Kokyu addresses the challenge that no *single* scheduling paradigm performs best in all environments, by enabling the configuration and empirical evaluation of multiple scheduling paradigms, including:

- **Static** scheduling strategies, *e.g.*, rate monotonic scheduling (RMS) [21],
- **Dynamic** scheduling strategies, *e.g.*, earliest deadline first (EDF) [21] and minimum laxity first (MLF) [6], and
- **Hybrid static/dynamic** scheduling strategies, *e.g.*, maximum urgency first (MUF) [6] and RMS+MLF [22].

This paper focuses on the hybrid static/dynamic MUF [6] and RMS+MLF [22] strategies, which were demonstrated in our recent work [2] to be preferable in COTS-based middleware when the total system load is infeasible but the critical subset of that load is still feasible. Specifically, both MUF and RMS+MLF were able to (1) *partition* critical and non-critical resource utilization using static mechanisms (such as thread priorities in COTS-based systems), and then (2) dynamically schedule single *operations*[3] within one [22] or more [6] of the partitions. The results in this paper illustrate how the Kokyu framework can provide adaptability across product families, operating systems, and most importantly environmental conditions, while preserving the rigorous scheduling guarantees and testability offered by prior work on statically scheduled CORBA operations [10, 23, 24] and multi-paradigm scheduling [2].

[2] Kokyu is a Japanese word meaning literally "breath", but also implying timing and coordination.

[3] We term the short-lived computation performed each time an event is pushed to a component an *operation*.

### 1.3 Performance Differences Between Similar Strategies

A crucial question for developers of mission-critical DRE systems is whether the differences *in policy alone* between alternative scheduling strategies are sufficient to distinguish which strategy (or sequence of alternative strategies) should be used *in practice* under a given set of system and environmental conditions. This paper presents empirical evidence that policy alone is *not* sufficient. Careful prototyping, modeling, analysis, and optimization of the actual behavior of the enforcement mechanisms themselves is therefore necessary to ensure good choices.

The work described in this paper is motivated by empirical results from our work on multi-paradigm scheduling [2]. Our results revealed differences in the performance of two scheduling strategies, RMS+MLF and MUF, that might have been expected to perform more similarly under the conditions of the experiment. Specifically, in terms of policy alone, the RMS+MLF and MUF strategies might not be expected to show meaningful differences in actual real-time performance. Our reasoning is presented below:

1. Under the experimental conditions described in Section 3, the sequence of resource requests for both critical and non-critical operations was identical in our experiments under the RMS+MLF and MUF scheduling strategies.
2. The pseudo-random sequence of jitter added to critical and non-critical operations was identical for RMS+MLF and MUF.
3. Since both RMS+MLF and MUF monotonically prioritize critical requests ahead of non-critical ones, the availability of the CPU to non-critical processing, as a function of time, should have been identical.
4. Since the RMS+MLF and MUF strategies both use laxity to order non-critical requests in a lowest-priority queue, the order in which non-critical requests were serviced should have been identical for the two strategies as well.

Clearly, policy alone is insufficient to explain the results we saw in our recent work with Kokyu. These results indicate the need to consider how that policy model might be extended to account for variations in the performance due to properties of the mechanisms used to implement the policies. This paper therefore presents new analysis of data obtained from our earlier experiments, and identifies newly discovered correlations within those data. For future work we offer a plausible model for the observed behavior of the RMS+MLF and MUF scheduling strategies and suggest additional hypotheses and experiments as to verify that model.

### 1.4 Paper Organization

The remainder of this paper is organized as follows: Section 2 describes the application, middleware, OS, and hardware configurations that comprise the open experimentation platform used for our empirical studies; Section 3 summarizes the experimental design factors relevant to the performance differences between scheduling strategies in our previous work; Section 4 presents new findings based on further analysis of the data; Section 5 summarizes our observations and makes recommendations based on our results, identifies a plausible model for variability in the scheduling strategies, and suggests hypotheses and experiments for future work; Section 6 compares our research on Kokyu with related work; Finally, Section 7 offers concluding remarks.

## 2 Open Experimentation Platform

This paper focuses on experiments conducted under the Adaptive Software Flight Demonstration (ASFD) program [25] on a mission-critical system that is representative of an important class of DRE systems: *the operational flight program (OFP) in an avionics mission computing system.* An OFP manages sensors and operator displays, navigates the aircraft's course, and controls on-board equipment. The avionics system used for those experiments consisted of OFP components hosted on the *Bold Stroke* domain-specific middleware infrastructure, which in turn is built using the distribution middleware capabilities and common middleware services provided by the TAO Real-time CORBA ORB.

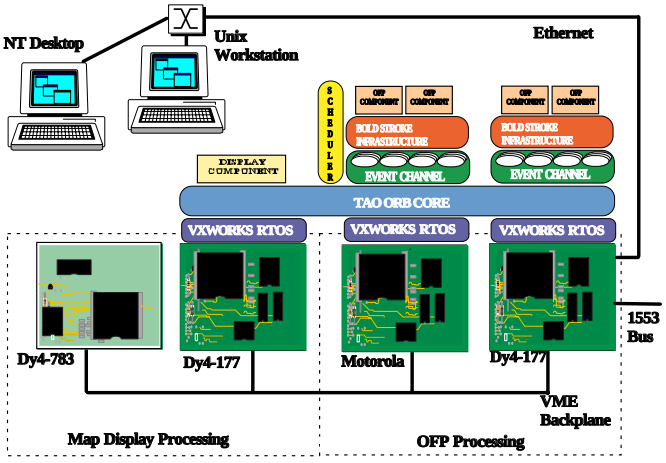


Figure 1: Open Experimentation Platform Hardware and Software

Figure 1 [2] illustrates the following OS/hardware, middleware, and application layers of the open experimentation plat-

form:[4]

**COTS OS/hardware.** The COTS hardware and operating system used in the experiments described in Section 3 consisted of a commercial VME-64 chassis with four commercial processor cards, a desktop computer running Windows NT 4.0 used for data gathering and visualization, and a portable UNIX workstation used to load executable programs onto the boards in the VME chassis and as a file server for a digital map display.

Two COTS processor cards, a Dy4-783 and a Dy4-177, performed the map display function. The Dy4-783 card had a memory-mapped display processor and the Dy4-177 card hosted an application component that ran the map display algorithms. The OFP system was distributed across the remaining two processor cards. The first system card was a 200 MHz, PowerPC 604, Motorola card, which ran the experimental system on the VxWorks [26] 5.3.1 real-time operating system. The second system card was a 100 MHz, PowerPC 603, Dy4-177 card. This card contained a MIL-STD-1553 MUX bus interface card and the Ethernet interface for the VME chassis. All external communication, *e.g.*, over the 1553 bus to connected sensors and actuators (called *remote terminals*) in the aircraft, or over the VME backplane to diagnostic and debug systems, went through this card. This card also controlled timing for frame sequencing and display updates, upon which operation rates on the Motorola card depended.

**DOC middleware.** The COTS distributed object computing middleware used for the ASFD demonstration was based on the TAO 1.2 implementation of Real-time CORBA [10, 9]. The TAO Real-time Event Channel [11] is a publish/subscribe service that mediates communication between components acting as proxies for (1) remote terminals that interact with the physical environment and (2) the operations that process the data. Sensor proxies flush relevant data to a Bold Stroke *replication service* that propagates the data between endsystems. The sensor proxies then *push* events through the Real-time Event Channel to the processing operations. Operation deadlines in the experimental system correspond to the points in time when their respective output values must be delivered and flushed to the replication service.

The Kokyu framework provides scheduling and dispatching services to TAO's Real-time Event Channel. Kokyu is responsible for (1) isolating critical processing from non-critical processing and (2) making the remaining CPU time available to non-critical processing. Kokyu provides these services via a scheduling strategy with which it is configured to (1) assign priorities to operations and (2) to specify the queueing discipline used at each priority level. By configuring TAO's Real-time Event Channel according to the specified set of priorities and queue disciplines, the middleware services described above enforce the mission computing system's real-time QoS assurances and performance.

**Bold Stroke domain-specific middleware.** The open experimentation platform for our work is based on the Bold Stroke domain-specific middleware [23, 24]. Bold Stroke uses COTS hardware and middleware to produce a standards-based component architecture for military avionics mission computing capabilities, such as navigation, data link management, and weapons control. A driving objective of Bold Stroke is to support reusable product-line applications, leading to a highly configurable application component model and supporting reusable middleware services, such replication and persistence services.

Bold Stroke has been developed and deployed using DOC middleware components and services based on the TAO Real-time ORB, the TAO Real-time Event Channel, and the Kokyu framework. Bold Stroke uses TAO's Real-time Event Channel atop the TAO ORB to communicate between components (1) on the same endsystem and (2) distributed across different endsystems. The Kokyu scheduler maintains information required for priority-preserving dispatching, which in the experimental framework described in Section 3 was performed in dispatching queues within the TAO Real-time Event Channel.

**OFP application.** The OFP application used as the basis of our multi-paradigm scheduling experiments provides avionics mission computing capabilities for an AV-8B (Harrier) aircraft. It is a distributed OFP implemented in C++ using the Boeing AV-8 Open Systems Core Avionics Requirements airframe [27] and the Boeing Bold Stroke domain-specific middleware. All major OFP components were implemented as periodically invoked operations, executed by event consumers. Each operation belongs to one of two equivalence classes:

- **Hard real-time (HRT) for critical operations**—Critical operations in the HRT class are those whose failure to meet any given deadline has potentially significant consequences for the correctness of the application.
- **Soft real-time (SRT) for non-critical operations**—Deadline success for the non-critical SRT operations is desirable but not strictly mandatory.

There were five pre-defined rates of execution in the system: 40 Hz, 20 Hz, 10 Hz, 5 Hz, and 1 Hz. Each operation ran at one of these rates. For the ASFD open experimentation platform, new 20 Hz SRT functions were added to the OFP, including routes and steering components, as well as a digital map display.

[4]This platform, and the studies conducted on it, were supported under the Adaptive Software Flight Demonstration (ASFD) program hosted by the Boeing Phantom Works Open Systems Architecture organization. This work was administered by the Embedded Systems Branch of the Information Directorate, Air Force Research Labs (AFRL), Wright-Patterson Air Force Base, Dayton, Ohio. Portions of the TAO ORB and the Bold Stroke open experimentation platform were developed under support from DARPA ITO.

# 3 Relevant Experimental Characteristics

This section outlines experimental factors relevant to analysis and modeling of the observed performance differences between Maximum Urgency First (MUF) [6], and Rate Monotonic Scheduling (RMS)+Minimum Laxity First (MLF) [22] under representative environmental conditions with varying *load* and *load jitter* on the open experimentation platform described in Section 2. The remainder of this section describes the new hypotheses we investigated for this paper, the variables that were controlled, and the variables that were measured in our studies.

## 3.1 Hypotheses

The hypotheses explored in these studies are shown in Table 1. This table also notes how we conducted our analysis to evaluate each hypothesis.

| Hypothesis | Analysis |
|---|---|
| The efficiency and effectiveness of each scheduling strategy are sensitive to *environmental* factors, *i.e.*, load and load jitter. | We examine both efficiency and effectiveness across widely varying load and load jitter conditions. |
| Performance differences between similar scheduling strategies may correlate more strongly with mechanism-level factors than policy-level factors. | We examine performance of similar strategies in conditions under which behavior is expected to be similar by policy alone. |
| Performance differences show meaningful correlation to a plausible model for mechanism-level behavior. | We compare fine-grain differences in performance data to differences in mechanisms and their plausible responses to variations in environmental conditions, *i.e.*, load and load jitter. |

Table 1: Hypotheses Studied and Analysis Approaches

To test these hypotheses through study of the empirical differences between the similar RMS+MLF and MUF hybrid static/dynamic scheduling strategies, we examined detailed operation dispatching success and failure data collected on the experimentation platform described in Section 2. The data came from identical trials using each of the following canonical scheduling strategies:

- RMS [21], which is a purely static strategy that assigns priorities in rate order and manages requests at each priority level in first-in-first-out (FIFO) order. We examined RMS performance for comparison as it gives insight into the bounds of sensitivity to increasing load and load jitter.
- MUF [6], which is a hybrid static/dynamic strategy that assigns static priorities by operation criticality, and schedules within each static priority by minimum laxity.
- RMS+MLF [22], is also a hybrid static/dynamic strategy, which first schedules critical operations according to rate and then non-critical operations at lower priority according to laxity.

We selected these strategies for further analysis since our earlier work [2] showed them to be the most applicable to OFP application requirements to support both hard real-time (HRT) and soft real-time (SRT) operations under a range of load and load jitter conditions.

## 3.2 Controlled Variables

To manage the effects of varying load and load jitter in the more diverse operating environments in which they are being asked to run, many next-generation DRE systems must satisfy resource demands that

1. Vary overall at longer time-scales across a series of stable epochs of operation and
2. Produce different degrees of jitter in invocation-to-invocation demands across shorter time-scales within each epoch.

To model variation in both load and load jitter imposed by these types of demands, the experiments on which the analysis presented here is based added operations to a sequence of twelve epochs of operation, each representing a distinct *operating region* [4] numbered 0–11, as shown in Figure 2 [2]. In this section, we summarize the experimental characteris-

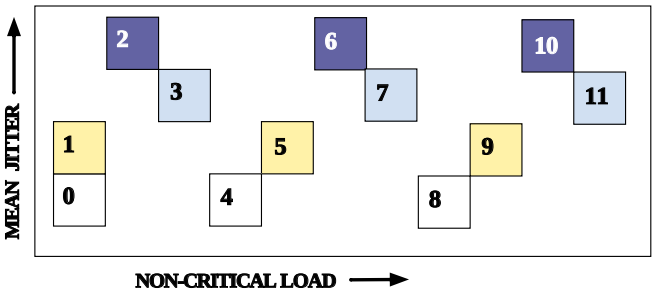


Figure 2: Operating Regions

tics of these operating regions to characterize the observed differences between the RMS+MLF and MUF scheduling strategies.

As we discussed in Section 1.3, the following experimental characteristics are relevant to examine the differences in performance between the RMS+MLF and MUF scheduling strategies:

- The sequence of resource requests for both critical and non-critical operations

- The pseudo-random sequence of jitter added to critical and non-critical operations
- The relative ordering of critical and non-critical request partitions and
- The ordering of non-critical operations.

We now consider each of these characteristics in turn:

**Sequence of resource requests.** Resource requests were made at each of the following rates: 20 Hz, 10 Hz, 5 Hz, and 1 Hz. The phasing of application operations was organized to reduce contention for the CPU overall, and to provide regular windows of low contention for the CPU, in which to extract collected data without interfering with the experiment itself. Figure 3 [2] shows the resulting framing of operations in the executing OFP.

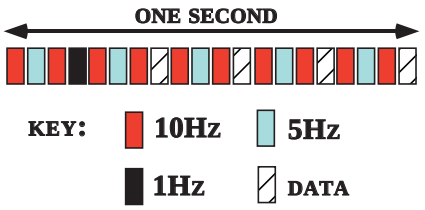


Figure 3: Framing of Operation Requests and Metrics Data Extraction Points

In addition to the fixed OFP operations, which were present and active in each operating region, we introduced chains of additional 20 Hz SRT navigation route leg[5] updates to each operating region. Computing route legs was implemented with each route segment successfully completed requesting the next segment, up to the length of the chain. We varied the length of the request chain to move from lowest to highest *fundamental* non-critical load. We did this incrementally from region 1 to region 11, while keeping the fundamental critical load constant across operating regions. We kept the non-critical load the same in region 0 and region 1 to ensure that we compared the effects of two different levels of jitter with no change in fundamental load in at least one case.

The key observation regarding the analysis presented in this paper is that both the RMS+MLF and MUF scheduling strategies used the same framing sequence (shown in Figure 3) and the same sequence of load. The same load in both critical and non-critical operations was therefore offered for each strategy, so we can meaningfully compare the performance differences between the strategies.

**Pseudo-random jitter sequence.** To examine the effects of (1) varying levels of load jitter across similar fundamental loads and (2) similar levels of jitter across varying non-critical loads, we added an additional HRT event consumer to the second Dy4-177 card at each of the following rates: 10 Hz, 5 Hz, and 1 Hz HRT. In these experiments, the additional operations acted as surrogates for the workload variation that would normally be associated with a distributed production OFP in a variable environment. The CPU utilization by these additional HRT event consumers was randomized across a given range in each operating region, with the range of variation cycling every four regions through the following:

1. 0 msec (lowest mean and lowest variance)
2. 0–5 msec (medium-low mean, medium variance)
3. 5–10 msec (highest mean, medium variance)
4. 0–10 msec (medium-high mean, highest variance)

Table 2 summarizes the HRT execution variability added to each operating region. *Total* variability was thus lowest in regions 0, 4, and 8, higher in regions 1, 5, and 9, higher still in regions 3, 7, and 11, and highest in regions 2, 6, and 10. The *range* of variability was lowest in regions 0, 4, and 8, was comparable in odd-numbered regions, and was highest in regions 2, 6, and 10.

| Regions | Variable HRT Execution |
|---|---|
| 0,4,8 | 0 msec |
| 1,5,9 | 0 to 5 msec |
| 2,6,10 | 5 to 10 msec |
| 3,7,11 | 0 to 10 msec |

Table 2: Load Jitter For Each Operating Region

The key observation for the analysis presented in this paper is that the execution time variability within each range was implemented as a pseudo-random sequence initialized with the same seed for each strategy. The load and load jitter were therefore identical for the RMS+MLF and MUF strategies.

**Relative ordering of request partitions.** By policy, both the RMS+MLF and MUF scheduling strategies (1) partition critical operations from non-critical operations, and (2) schedule operations in the critical partition in preference to operations in the non-critical partition. Moreover, these policies were implemented using thread priorities from a preemptive multi-tasking real-time OS (VxWorks) as described in Section 2. The relative ordering of request partitions was therefore also the same for RMS+MLF and MUF.

**Ordering of non-critical operations.** By policy, both RMS+MLF and MUF place all non-critical operations in a lowest-priority queue managed by operation laxity. The same values for relevant operation characteristics (*i.e.*, period and execution time) were used for each operation under each strategy. The non-critical operations were therefore handled the same way under the RMS+MLF and MUF strategies.

[5] A route leg is a segment of a navigation route computed in one operation invocation.

### 3.3 Measured Variables

Section 3.2 showed that at a policy level the RMS+MLF and MUF strategies were indistinguishable under the *controlled* variables of the experiment. Below, we examine the *measured* variables of the experiment, which distinguish the behavior of the RMS+MLF and MUF strategies. We also offer insights into the possible reasons for those differences in Section 5.

To measure the response of each strategy to the varying load and load jitter described in Section 3.2, we instrumented the application and middleware using lightweight, high-resolution time stamps to characterize system behavior. We focus here on the missed and made operation deadlines.

The missed and made operation deadlines offer two kinds of information. First, we can assess the real-time *effectiveness* of a scheduling strategy by measuring the number (and category) of deadlines made. Our definition of real-time effectiveness differs from conventional notions of throughput: within a sample, each SRT operations is counted only if it made its deadline, and no HRT operation deadlines were missed in the entire sample. The higher the total number of SRT deadlines that were made without missing any HRT deadlines, the higher the level of performance of the strategy. Second, we can assess the real-time *efficiency* of the scheduling strategies by also examining the *fraction* of SRT operation deadlines that were made. Efficiency as we define it is simply the SRT effectiveness divided by the total number of SRT operations, *i.e.*, the *offered load*. The higher the fraction of SRT operation deadlines that were made, again without missing any HRT deadlines, the higher the efficiency of a strategy.

In general, the measured efficiency is only important when the offered load has significance, *i.e.*, when the SRT and HRT operations together exceed the feasible utilization bound for the CPU, as is seen in operating regions 7 through 11. Otherwise, the effectiveness measure is sufficient to distinguish the behaviors of the scheduling strategies. The distribution of efficiency values, as shown in Section 4.2, provides insight into the finer-grained behavior of each strategy. In aggregate, this information can be used to profile the response of a strategy to the offered load and load jitter in each operating region.

## 4 Empirical Results

We now present results from new analysis of the trials described in Section 3, using the open experimental platform described in Section 2. Specifically, we systematically examine the hypotheses described in Table 1 and note how our observations do or do not support the hypothesis in each case. We thus empirically evaluate the differences between the RMS+MLF and MUF strategies in practice. In the process, we also uncover insights relevant to developers of mission-critical DRE systems.

### 4.1 Efficiency and Effectiveness of SRT Operation Dispatching

**Hypothesis.** The efficiency and effectiveness of each scheduling strategy are sensitive to *environmental* factors, *i.e.*, load and load jitter.

**Overview of the analysis.** To evaluate this hypothesis, we examine how the efficiency and effectiveness of each strategy varied with changing load and load jitter. In addition to RMS+MLF and MUF (which are similar hybrid static/dynamic strategies), we also examine the behavior of the canonical RMS static strategy for purposes of comparison.

**Synopsis of results.** Figure 4 shows the relative sensitivity of RMS, RMS+MLF, and MUF to variations in load and load jitter, such as might be expected to occur from variations in a next-generation DRE system's environment. The vertical axis in Figure 4 shows an average weighted SRT performance function, over the different operating regions described in Section 3.2, which are shown on the horizontal axis. For each sample, a value of zero was assigned if any HRT deadlines were missed in that sample, or otherwise the value was the number of SRT deadlines made. An average was then taken over those values in each operating region, for each strategy.

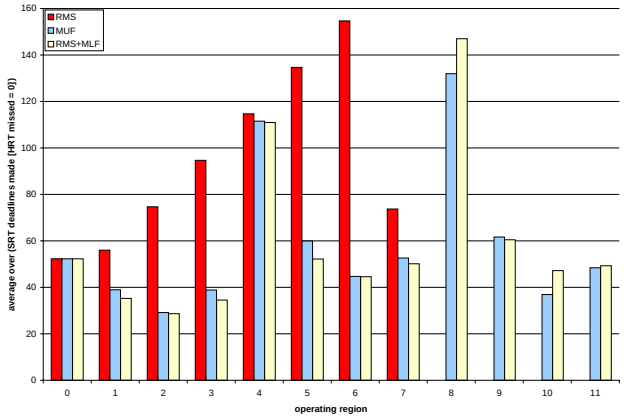


Figure 4: Relative Performance of RMS, RMS+MLF, and MUF

The RMS strategy showed increasing performance across operating regions 0 through 6. It then rapidly decreased through operating region 7 to minimal performance in operating regions 8 through 11. The RMS+MLF and MUF strategies both showed cyclic patterns of performance across operating regions 0 through 11, with the cycle spanning four operating regions.

In addition to the SRT performance differences indicated in Figure 4, we note that while RMS offered the best SRT per-

formance in operating regions 0 through 7, in operating region 7 RMS showed 5 samples with missed HRT deadlines (in operating regions 8 through 11 RMS showed missed HRT deadlines in each sample). We also note that while MUF performed slightly better than RMS+MLF in operating region 9, MUF had a single sample with a single missed HRT deadline late in the operating region.

**Analysis of results.** Figure 4 shows strong correlation between the performance of scheduling strategies and environmental factors. More interestingly, the results discussed above correlate with different factors for RMS than the RMS+MLF or MUF, and correlate with the same factor for RMS+MLF and MUF. In particular, the RMS performance correlated strongly with varying total load, but had no observable correlation with the variations in load jitter. RMS+MLF and MUF performance showed strong correlation with load jitter, but only weak correlation (*i.e.*, amplitude of the performance cycle) with variations in system load. Moreover, the *difference* in performance between RMS+MLF and MUF varied meaningfully as a function of the additional jitter in resource request execution times, although the offered load and load jitter of resource requests were kept the same for the two strategies in each operating region of our experiments.

**Summary.** The results above support the hypothesis that the efficiency and effectiveness of each scheduling strategy are sensitive to *environmental* factors, and refine the hypothesis with respect to which factors are the most relevant to each of the strategies studied. In the ASFD program, we collaborated with DRE system developers to identify canonical scheduling strategies and important environmental factors and to examine the sensitivities of those in realistic operating environments. Furthermore the results presented above identify which strategies performed best under different environmental conditions. In Section 5 we extend these observations and make observations for mission-critical COTS-based DRE systems in general, to ensure environmental factors are adequately addressed in selecting middleware scheduling strategies.

## 4.2 Mechanism-Level Correlation

**Hypothesis.** Performance differences between similar scheduling strategies may correlate more strongly with mechanism-level factors than policy-level factors.

**Overview of the analysis.** To evaluate this hypothesis, we examine how differences in mechanism-level factors correlated with differences in the performance of each strategy. We focus on the extent to which each strategy meets non-critical deadlines under different conditions of load and load jitter where policy differences do not distinguish the strategies. As we argued in Section 1.3, there is no meaningful correlation between policy in RMS+MLF and MUF and the observed performance differences, under the experimental conditions described in Section 3.2. We therefore focus on whether meaningful correlation can be established between differences in the mechanisms used to implement RMS+MLF and MUF and the observed performance.

**Synopsis of results.** Figure 5 shows the weighted fraction of SRT deadlines made for each sample in operating region 8. We note two main characteristics of operating region 8:

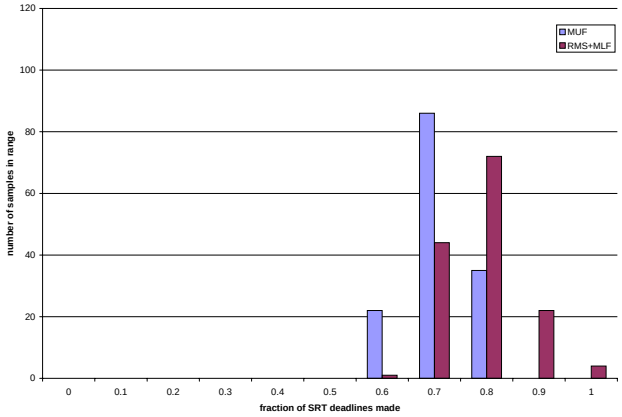


Figure 5: Weighted SRT Fraction in Operating Region 8

1. As with each of the operating regions 7 through 11, the total load is above the feasible threshold, while the critical load remains feasible. Figure 5 illustrates two important similarities between performance of the RMS+MLF and MUF strategies under these conditions. First, both strategies achieved high *levels* of SRT deadline success, with no missed HRT deadlines.
2. The *distribution* of values is similar and is reasonably well bounded in each case. MUF showed slightly more variation in its SRT performance values, and consistently exhibited slightly lower levels of success than RMS+MLF.

Figure 6 shows the weighted fraction of SRT deadlines made for each sample in operating region 9. Operating region 9 had slightly higher total load than region 8, but more importantly had an additional medium-low (0-5 msec) level of load jitter. Under these conditions, both strategies showed lower average levels of SRT deadline success, and a wider range of values overall. Interestingly, while the overall ranges of values were similar, MUF showed a much more continuous distribution of values than RMS+MLF. Interestingly, while MUF had slightly lower minimum and maximum values, its performance on average was better than that of RMS+MLF. We also note a single sample in which MUF missed one HRT deadline, toward the end of the time spent in operating region 9.

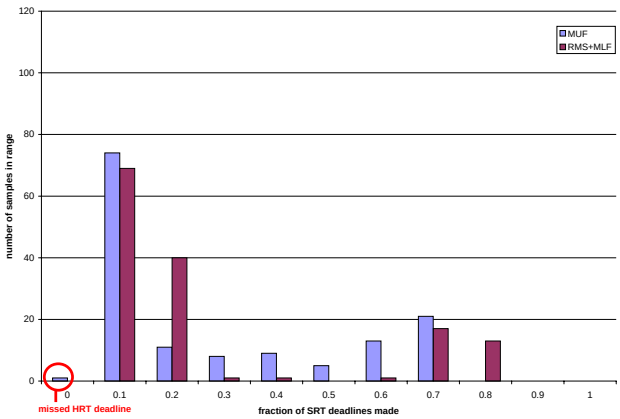


Figure 6: Weighted SRT Fraction in Operating Region 9

**Analysis of results.** In each of the graphs above, the level and distribution of SRT performance values distinguish the strategies in ways that map well to mechanism-level differences. For MUF, the ordering of HRT operations in a dynamic queue necessarily adds overhead compared to the best-case performance of multiple FIFO queues at different thread priorities, which is how RMS+MLF manages its HRT operations. As discussed previously, the policy differences in the absence of other factors indicate the type of similar performance seen in Figure 5, albeit with a downward shift in the MUF values due to mechanism-level dynamic scheduling overhead.

Consider the pacing of operation dispatch requests under conditions of little or no load jitter, and the handling of requests in priority order. These factors offer evidence that in Figure 5 we do in fact see the best-case mechanism-level performance of the RMS+MLF strategy, where in each 20Hz frame the highest priority operations all run to completion, then the next highest priority, and so forth. In figure 6, we see evidence that the performance differences indicates pulses of preemption overhead, contributing to higher overhead on average in RMS+MLF. We note especially the reasonably periodic and bi-modal distribution of the RMS+MLF performance values. This suggests that the medium-low level of load jitter in operating region 9 resulted in non-optimal arrival patterns with lower priority requests arriving immediately before a higher priority request. In practice systems are architected to avoid realizing the worst case request arrival patterns which result in a critical instant [21]. The degree to which one succeeds is affected by variations in job arrival and processing times.

**Summary.** The results above support the hypothesis that performance differences between similar scheduling strategies may correlate more strongly with mechanism-level factors than policy-level factors. We focused our attention on implementation mechanisms of scheduling strategies in the ASFD operating environment. In doing so, we correlated different mechanism-level responses to environmental factors with real-time behvaior. We found that benchmarking canonical scheduling strategies in the ASFD operating environment was essential to evaluate our middleware scheduling implementation. Section 5 describes recommendations to developers of next-generation DRE systems based on these observations.

## 4.3 Correlation with a Plausible Model

**Hypothesis.** Performance differences show meaningful correlation to a plausible model for mechanism-level behavior.

**Overview of the analysis.** To evaluate this hypothesis, we compare fine-grain differences in performance data to differences in mechanisms and their plausible responses to variations in environmental conditions, *i.e.*, load and load jitter. In particular, we examine weighted SRT performance in operating regions 8 through 11, each of which represents a canonical case of load and load jitter conditions.

**Synopsis of results.** Figure 7 shows the weighted fraction of SRT deadlines made for each sample in operating region 10. A similar relationship between high jitter and low jitter to

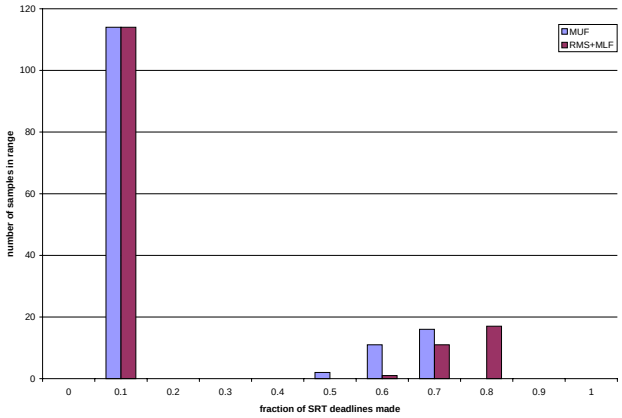


Figure 7: Weighted SRT Fraction in Operating Region 10

performance of the strategies is shown in operating regions 8 and 10. MUF values are offset slightly downward in region 10, and the overall distributions of values are narrow, albeit at lower levels than in region 8.

Figure 8 shows the weighted fraction of SRT deadlines made for each sample in operating region 11. A similar relationship between medium-low and medium-high jitter to performance of the strategies is shown in operating region 11. MUF values are distributed more continuously than those of RMS+MLF.

**Analysis of results.** In each of the graphs above, and those shown in Section 4.2, the performance differences between the strategies correlate strongly with the expected response of

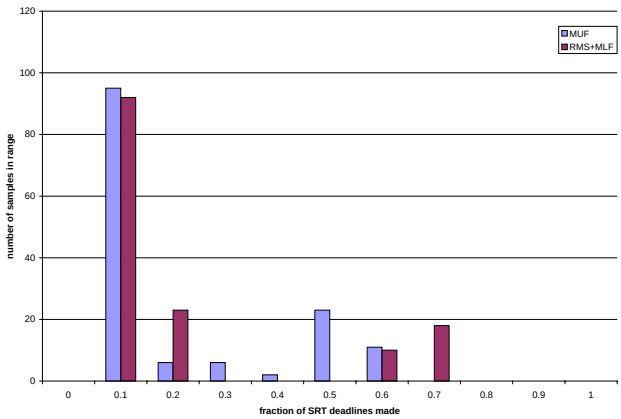


Figure 8: Weighted SRT Fraction in Operating Region 11

mechanism-level factors to the load and load jitter conditions in each operating region. Specifically, it appears that a phasing effect in the pacing of HRT operations due to jitter is responsible for these results, with RMS+MLF suffering little preemption overhead in the zero-jitter regions (0,4,8), higher preemption overhead in the regions with medium-low or medium-high jitter (1,3,5,7,9,11), and then low preemption overhead again in the high-jitter regions (2,6,10).

**Summary.** The results above support the hypothesis that performance differences show meaningful correlation to a plausible model for mechanism-level behavior. Our observations suggest that intermediate levels of jitter introduced experiments described in Section 3 led to greater overhead of the RMS+MLF mechanisms. We attribute this effect to greater preemption and context switching overhead as a result of a form of phasing in the arrival of HRT dispatch requests–as the jitter was increased or decreased away from medium levels, RMS+MLF showed less distribution in its SRT deadline success values overall. In Section 5, we offer recommendations based on our experiences establishing the empirical basis for (and constructing) this model, identify key elements of the model, and suggest further hypotheses and experiments to validate the model as future work.

# 5 Lessons Learned and Future Research

Below, we present key observations and recommendations based on our empirical results from Section 4. These observations and recommendations apply both to the particular avionics mission computing application we have studied and to a larger family of mission-critical DRE systems. We also describe future work to validate a model for scheduling variability based on our observations.

## 5.1 Summary of Lessons Learned

As we noted in Section 1.1, the use of COTS operating systems and middleware is highly desirable for building and maintaining mission-critical DRE systems in a cost-effective manner. However, COTS software cannot simply be applied to applications with specialized processing requirements, without careful empirical study of the implications of the specific policies and mechanisms employed within those systems. In particular, real-time applications have stringent requirements that may vary with environmental stimuli, system load or operator intervention.

Fundamental difficulties in providing predictable behavior of complex systems when they are under moderate to heavy loads can compound this problem. Individual policies and mechanisms, such as those for the scheduling strategies we have studied, may exhibit differing overheads and sensitivities to environmental conditions. However, with judicial use of COTS building blocks and armed with a core understaning of the underlying mechanisms it is possible to build high-performance, real-time applications. Our research has identified several key areas where developers can improve system behavior and predictability:

• **Observation.** The efficiency and effectiveness of each scheduling strategy are sensitive to *environmental* factors.

• **Recommendation.** Middleware researchers and DRE system developers should collaborate to ensure that (1) the environmental sensitivities of particular scheduling strategies are well characterized in realistic operating environments, and (2) the factors present in a particular DRE system's particular operating environment are used to choose between scheduling strategies for that system.

• **Observation.** Performance differences between similar scheduling strategies may correlate more strongly with mechanism-level factors than policy-level factors.

• **Recommendation.** *Middleware researchers* should focus attention on mechanism-level implementation details of scheduling strategies in COTS-based environments, to offer (1) robust real-time assurances and (2) control over trade-offs for optimization. Moreover, *developers* of COTS-based DRE systems should benchmark their scheduling strategies in operationally meaningful environments, to evaluate the extent to which a particular middleware implementation has addressed these issues.

• **Observation.** Performance differences between the RMS+MLF and MUF strategies show meaningful correlation to a plausible model for mechanism-level behavior. Specifically, our results suggest that preemption and queue overhead effects are important factors for modeling the performance differences between these strategies.

• **Recommendation.** Middleware researchers and developers of COTS-based mission-critical DRE systems should collaborate to construct, extend, and empirically evaluate detailed models for *mechanism-level* scheduling and dispatching behavior in COTS hardware and software environments. Moreover, these models should be evaluated rigorously in terms of critical assurances and non-critical performance. In particular, as preferred models emerge from this effort, additional attention should be paid to refining, cross-validating models across applications and domains, and unifying models where possible to attain a more coherent picture of COTS-based real-time QoS management. This larger-scale effort will in turn lead to higher-quality models and techniques for managing real-time QoS in mission-critical COTS-based DRE systems.

### 5.2 Future Work

Our experimental results in Section refempirical:results validated the ability of scheduling frameworks to adapt strategies to changing load and operation mode. However, they also raised issues related to the sensitivity of the mechanisms used in the operational environment. It is common practice when designing real-time systems to select a strategy based on worst-case analysis. However, our results show that this is not necessarily optimal nor desirable. In practice, different policies and their implementation exhibit varying overheads and sensitivities to environmental conditions.

For example, in theory, we would expect that partitioning the HRT and SRT operations by scheduling priority would protect the critical operations, thus ensuring they always meet their deadlines. This is not the case, however, as shown by the missed critical deadline in operating region 9 for MUF (see Section 4.2). We would also suspect that given a feasible schedule for the critical tasks the use of a non-preemptive dispatching strategy for the critical operations as provided by the MUF implementation would result in overall lower overhead as the overhead of preemption is not necessary nor will queue depths be large for our structured environment. While this appears to be true for operating regions 0 through 7, under heavy loads it breaks down.

These observations lead to the identification of several key aspects of the system that contribute to variability of the results:

- Operation scheduling parameters
- Preemption model based on thread priorities and request arrival
- Queueing discipline for ordering within each thread priority level and
- Mechanism overhead in response to the above factors.

We propose these factors as the basis for a mechanism-level model for variability of different scheduling strategies, which we will evaluate as future work. Specifically, we will examine the following hypotheses regarding the MUF and RMS+MLF strategies–and the experiment following each hypothesis–to validate this middleware scheduling model:

• **Hypothesis 1.** *Overhead differences in the middleware and OS can be shown to account for the observed latency differences between the strategies.* To validate this hypothesis, we are devising experiments to measure and compare key sources of overhead, such as queue depths, queue ordering cost, thread context switch cost, degree of preemption, in the middleware and OS under experimental conditions comparable to those described in Section 3.

• **Hypothesis 2.** *The question of whether (a) thread preemption and context switching overhead or (b) overhead from ordering operations in a dynamic queue is greater under each distinct set of load and load jitter conditions, can be shown to account for the performance differences between the RMS+MLF and MUF strategies under those conditions.* To validate this hypothesis, we are devising experiments to measure and compare the complete preemption timeline for all operations to the measured middleware and OS overhead factors under comparable experimental conditions to those described in Section 3.

## 6 Related Work

Distributed real-time and embedded (DRE) computing is an emerging field of study. An increasing number of research efforts are focusing on end-to-end quality of service (QoS) properties, such as timeliness, by integrating QoS management policies and mechanisms, *e.g.*, real-time scheduling into standards-based middleware, such as Real-time CORBA. Pioneering efforts are beginning to extend this field by providing meta-capabilities, such as configuration flexibility, reflection, and ultimately adaptation, while still meeting strict QoS assurances. This section describes representative work that is related to our Kokyu framework.

**Avionics platform research.** The following two branches of research are endeavoring to make QoS-managed system infrastructure a prevalent and reusable feature of avionics computing systems:

• **Avionics domain platform research.** Standardized avionics platforms, such as the ARINC Avionics Application Software Standard Interface (APEX) for Integrated Modular Avionics (IMA) [28], provide QoS assurances for systems in the avionics domain. McElhone [29] examines the question of how to support operations with soft real-time constraints and possibly long running or variable length computations, in canonical avionics-specific platforms, such as IMA.

• **Open systems avionics research.** Sharp, Doerr, *et al*. [23, 24] address the challenge of retaining key QoS assurances in avionics systems, while achieving improvements in modularity, reuse, cycle times, and cost across families of flight software applications. The Bold Stroke avionics domain-specific middleware described in Section 2 has emerged and evolved through that work. Our research on flexible and adaptive real-time scheduling and dispatching was conducted within the context of the Bold Stroke infrastructure, and has contributed to its evolution.

**CORBA-related QoS middleware research.** There is a growing body of work related to CORBA-based QoS middleware. We focus below on related CORBA middleware research efforts that address scheduling or other forms of adaptive QoS management.

• **Standard specifications.** The OMG Real-Time CORBA 1.0 [30] specification includes interfaces for an optional scheduling service that can be implemented readily using Kokyu's flexible scheduling and dispatching capabilities. We plan to release an implementation of this service built using the Kokyu framework. Emerging COTS middleware standards, such as Dynamic Scheduling Real-Time Common Object Request Broker Architecture (CORBA) 2.0 (DSRTCORBA) [31], as well as the non-CORBA Real-Time Specification for Java$^{TM}$ (RTSJ) [32], generalize the possible range of scheduler implementations, rather than specifying a particular scheduling approach. Kokyu offers a natural basis for reuse of policies and mechanisms in implementing schedulers and associated dispatching infrastructures for either of these standards.

• **BBN QuO.** The *Quality Objects* (QuO) distributed object middleware is developed at BBN Technologies [33]. QuO is based on CORBA and provides the following support for agile applications running in wide-area networks: (1) *run-time performance tuning and configuration* through the specification of *QoS regions*, behavior alternatives, and reconfiguration strategies that allows the QuO run-time to adaptively trigger reconfiguration as system conditions change (represented by transitions between operating regions) and (2) *feedback* across software and distribution boundaries based on a control loop in which client applications and server objects request levels of service and are notified of changes in service. We have integrated Kokyu into the QuO framework, as described in [4].

• **UCSB Realize.** The Realize project at UCSB has developed an approach based on object migration and replication, to improve performance of soft real-time distributed systems [34, 35]. This approach constitutes a higher level of adaptive control for soft real-time QoS management, and is complementary to Kokyu. In particular, a system developer might apply Realize to provide soft real-time load balancing across endsystems, using the Kokyu framework to integrate scheduling and dispatching of both critical and non-critical load.

• **UCI TMO.** The Time-triggered Message-triggered Objects (TMO) project [36] at the University of California, Irvine, supports the integrated design of distributed OO systems and real-time simulators of their operating environments. The TMO model provides structured timing semantics for distributed real-time object-oriented applications by extending conventional invocation semantics for object methods, *i.e.*, CORBA operations, to include (1) invocation of time-triggered operations based on system times and (2) invocation and time bounded execution of conventional message-triggered operations. TMO, Kokyu, and TAO are complementary technologies because (1) TMO and Kokyu extend and generalize TAO's existing time-based invocation capabilities and (2) TAO provides a configurable and dependable connection infrastructure needed by the TMO CNCM service.

**Non-CORBA QoS research.** In addition to CORBA-related QoS middleware research, our work on Kokyu is also related to the following QoS research conducted outside CORBA:

• **Utah CRM.** Regehr and Lepreau [37] propose the CPU Resource Manager (CRM), a middleware service for managing processor allocation using scheduling abstractions provided by commodity-off-the-shelf (COTS) operating systems. They examine different kinds of QoS reservations and propose a unifying low-level middleware abstraction layer to shield developers from accidental complexities produced by variations in scheduling abstractions at the operating system level. Our approach focuses on *encapsulation* of scheduling and dispatching policies, and providing flexible infrastructure to allow arbitrary composition of heuristics. Rather than enclosing a known set of common abstractions, our aim is to provide flexible support for diverse and possibly unanticipated combinations of scheduling requirements, mechanisms, and policies in middleware.

• **UCI RED-Linux Scheduling Framework.** Wang, *et al.* [38], at the University of California, Irvine, have proposed a general scheduling framework to unify three distinct kinds of scheduling approaches: *priority-based*, *time-based*, and *share-based*. They decompose scheduling behavior into policy (*allocator*) and mechanism (*dispatching*) components, which are similar to the Kokyu scheduling service framework. They have implemented the dispatching portion of this framework in their real-time extensions to the Linux kernel, called RED-Linux. While the RED-Linux approach to scheduling relies on special-purpose extensions to the OS kernel, our Kokyu framework relies only on commonly available OS features, such as preemptive thread priorities. Our dispatching mechanisms can therefore augment standards-based CORBA middleware and can perform effectively on a wide range of commonly available real-time and general-purpose OS platforms.

• **Feedback Control Scheduling.** One of the most important areas of related work is the pioneering research on

feedback control real-time scheduling (FCS), conducted by Stankovic, Lu, *et al.*, at the University of Virginia. They apply control theory to real-time scheduling [39, 40, 41, 42, 43, 44] for soft real-time systems, to reduce the number of missed deadlines at run-time. We consider the FCS work highly complementary to our efforts to model, and distinguish through empirical study, strategies for real-time assurance and performance optimization.

The primary difference between the FCS work and our Kokyu work is that they focus on controlling a single performance metric, the deadline performance of soft real-time tasks. Our research is aimed primarily at distributed rate-based systems where at least two classes of operations are present, and deadlines for the highest class must be assured before soft real-time performance is optimized. For example, FCS specifies a *miss ratio function* [42], which is comparable to the fraction of SRT deadlines made that is described in Section 3.3, though they measure the fraction of srt deadlines missed, rather than made. However, we then predicate this raw measure of soft real-time performance with whether *any* critical deadlines have been missed in each sample, to obtain a performance metric that considers multiple criticality levels.

# 7 Concluding Remarks

New increasingly non-deterministic types of processing, such as video and imaging [4], are being targeted for transition to existing mission-critical distributed real-time and embedded (DRE) systems. In recent work [2], we showed how Kokyu's ability to manage variations in execution load and load jitter through alternative scheduling strategies increases the applicability of these techniques to DRE systems built using COTS software architectures. This paper extended those results by further analysis that offers more exact guidance on choice of scheduling strategies to developers of mission-critical DRE systems. In particular, this paper presented new empirical evidence that the observed performance differences between the RMS+MLF and MUF strategies are due to mechanism-level effects in the face of load jitter, rather than policy-level differences in the capabilities of the scheduling strategies.

# Acknowledgments

We gratefully acknowledge the support and direction of the AFRL program manager for ASFD, Kenneth Littlejohn, and of Boeing Bold Stroke Principal Investigators Bryan Doerr and David Sharp. In addition, we would like to thank Greg Holtmeyer for his contributions to this research.

# References

[1] C. D. Gill, R. Cytron, and D. C. Schmidt, "Middleware Scheduling Optimization Techniques for Distributed Real-Time and Embedded Systems," in *Proceedings of the* $7^{th}$ *Workshop on Object-oriented Real-time Dependable Systems*, (San Diego, CA), IEEE, Jan. 2002.

[2] C. Gill, D. C. Schmidt, and R. Cytron, "Multi-Paradigm Scheduling for Distributed Real-Time Embedded Computing," *IEEE Proceedings Special Issue on Modeling and Design of Embedded Software*, Oct. 2002.

[3] C. D. Gill, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-Time CORBA Scheduling Service," *Real-Time Systems, The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware*, vol. 20, Mar. 2001.

[4] J. Loyall, J. Gossett, C. Gill, R. Schantz, J. Zinky, P. Pal, R. Shapiro, C. Rodrigues, M. Atighetchi, and D. Karr, "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications," in *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pp. 625–634, IEEE, Apr. 2001.

[5] D. A. Karr, C. Rodrigues, Y. Krishnamurthy, I. Pyarali, and D. C. Schmidt, "Application of the QuO Quality-of-Service Framework to a Distributed Video Application," in *Proceedings of the 3rd International Symposium on Distributed Objects and Applications*, (Rome, Italy), OMG, Sept. 2001.

[6] D. B. Stewart and P. K. Khosla, "Real-Time Scheduling of Sensor-Based Control Systems," in *Real-Time Programming* (W. Halang and K. Ramamritham, eds.), Tarrytown, NY: Pergamon Press, 1992.

[7] R. E. Schantz and D. C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," in *Encyclopedia of Software Engineering* (J. Marciniak and G. Telecki, eds.), New York: Wiley & Sons, 2002.

[8] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*. Reading, MA: Addison-Wesley, 1999.

[9] Object Management Group, *The Common Object Request Broker: Architecture and Specification, Revision 2.6*, Dec. 2001.

[10] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers," *Computer Communications*, vol. 21, pp. 294–324, Apr. 1998.

[11] T. H. Harrison, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," in *Proceedings of OOPSLA '97*, (Atlanta, GA), pp. 184–199, ACM, Oct. 1997.

[12] A. Gokhale and D. C. Schmidt, "Measuring and Optimizing CORBA Latency and Scalability Over High-speed Networks," *Transactions on Computing*, vol. 47, no. 4, 1998.

[13] F. Kuhns, D. C. Schmidt, C. O'Ryan, and D. Levine, "Supporting High-performance I/O in QoS-enabled ORB Middleware," *Cluster Computing: the Journal on Networks, Software, and Applications*, vol. 3, no. 3, 2000.

[14] C. O'Ryan, F. Kuhns, D. C. Schmidt, O. Othman, and J. Parsons, "The Design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware," in *Proceedings of the Middleware 2000 Conference*, ACM/IFIP, Apr. 2000.

[15] D. C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," *Journal of Real-time Systems, special issue on Real-time Computing in the Age of the Web and the Internet*, vol. 21, no. 2, 2001.

[16] A. B. Arulanthu, C. O'Ryan, D. C. Schmidt, M. Kircher, and J. Parsons, "The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging," in *Proceedings of the Middleware 2000 Conference*, ACM/IFIP, Apr. 2000.

[17] C. O'Ryan, D. C. Schmidt, F. Kuhns, M. Spivak, J. Parsons, I. Pyarali, and D. L. Levine, "Evaluating Policies and Mechanisms to Support Distributed Real-Time Applications with CORBA," *Concurrency and Computing: Practice and Experience*, vol. 13, no. 2, pp. 507–541, 2001.

[18] O. Othman, C. O'Ryan, and D. C. Schmidt, "An Efficient Adaptive Load Balancing Service for CORBA," *IEEE Distributed Systems Online*, vol. 2, Mar. 2001.

[19] N. Wang, D. C. Schmidt, O. Othman, and K. Parameswaran, "Evaluating Meta-Programming Mechanisms for ORB Middleware," *IEEE Communication Magazine, special issue on Evolving Communications Software: Techniques and Technologies*, vol. 39, Oct. 2001.

[20] A. Gokhale and D. C. Schmidt, "Optimizing a CORBA IIOP Protocol Engine for Minimal Footprint Multimedia Systems," *Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems*, vol. 17, Sept. 1999.

[21] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, vol. 20, pp. 46–61, Jan. 1973.

[22] J.-Y. Chung, J. W.-S. Liu, and K.-J. Lin, "Scheduling Periodic Jobs that Allow Imprecise Results," *IEEE Transactions on Computers*, vol. 39, pp. 1156–1174, Sept. 1990.

[23] D. C. Sharp, "Reducing Avionics Software Cost Through Component Based Product Line Development," in *Proceedings of the 10th Annual Software Technology Conference*, Apr. 1998.

[24] B. S. Doerr and D. C. Sharp, "Freeing Product Line Architectures from Execution Dependencies," in *Proceedings of the 11th Annual Software Technology Conference*, Apr. 1999.

[25] W.-P. A. F. B. Air Force Research Labs, "Adaptive Software Flight Demonstration (ASFD)." Delivery Order 003 of the WSSTS contract to The Boeing Company, number F33615-97-D-1155, 1999.

[26] Wind River Systems, "VxWorks 5.3." www.wrs.com/products/html/vxworks.html.

[27] T. B. Company, "Open Systems Core Avionics Requirement (OSCAR)." http://www.acq.osd.mil/osjtf/pdf/oscar.pdf.

[28] ARINC Incorporated, Annapolis, Maryland, USA, *Document No. 653: Avionics Application Software Standard Inteface (Draft 15)*, Jan. 1997.

[29] C. McElhone, "Soft Computations within Integrated Avionics Systems," in *Proceedings of the IEEE National Aerospace and Electronics Conference (NAECON 2000)*, Oct. 2000.

[30] Object Management Group, *Real-time CORBA Joint Revised Submission*, OMG Document orbos/99-02-12 ed., Mar. 1999.

[31] Object Management Group, *Dynamic Scheduling Real-Time CORBA 2.0 Joint Final Submission*, OMG Document orbos/2001-06-09 ed., Apr. 2001.

[32] Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull, *The Real-Time Specification for Java*. Addison-Wesley, 2000.

[33] J. A. Zinky, D. E. Bakken, and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems*, vol. 3, no. 1, pp. 1–20, 1997.

[34] V. Kalogeraki, P. M. Melliar-Smith, and L. E. Moser, "Dynamic Migration Algorithms for Distributed Object Systems," in *21st IEEE International Conference on Distributed Computing Systems (ICDCS)*, (Phoenix AZ), IEEE, Apr. 2001.

[35] V. Kalogeraki, P. M. Melliar-Smith, and L. E. Moser, "Dynamic Scheduling of Distributed Method Invocations," in *21st IEEE Real-Time Systems Symposium*, (Orlando, FL), IEEE, Nov. 2000.

[36] K. H. K. Kim, "Object Structures for Real-Time Systems and Simulators," *IEEE Computer*, pp. 62–70, Aug. 1997.

[37] J. Regehr and J. Lepreau, "The Case for Using Middleware to Manage Diverse Soft Real-Time Schedulers," in *Proceedings of the International Workshop on Multimedia Middleware (M3W '01)*, (Ottowa, Canada), Oct. 2001.

[38] Y.-C. Wang and K.-J. Lin, "Implementing A General Real-Time Scheduling Framework in the RED-Linux Real-Time Kernel," in *IEEE Real-Time Systems Symposium*, pp. 246–255, IEEE, Dec. 1999.

[39] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 2002, to appear.

[40] C. Lu, *Feedback Control Real-Time Scheduling*. PhD thesis, University of Virginia, Charlottesville, VA, May 2001.

[41] J. A. Stankovic, T. He, T. F. Abdelzaher, M. Marley, G. Tao, S. H. Son, and C. Lu, "Feedback Control Scheduling in Distributed Systems," in *The 22nd IEEE Real-Time Systems Symposium (RTSS '01)*, (London UK), Dec. 2001.

[42] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley, "Performance Specifications and Metrics for Adaptive Real-Time Systems," in *The 21st IEEE Real-Time Systems Symposium (RTSS '00)*, (Orlando FL), Dec. 2000.

[43] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Design and Evaluation of a Feedback Control EDF Scheduling Algorithm," in *The 20th IEEE Real-Time Systems Symposium (RTSS '99)*, (Phoenix AZ), Dec. 1999.

[44] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao, "The Case for Feedback Control Real-Time Scheduling," in *11th EuroMicro Conference on Real-Time Systems*, (York UK), June 1999.