Rate-Monotonic Scheduling in the Presence of Timing Unpredictability

Lei Zhou, Kang G. Shin
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, Michigan 48109-2122
{LZHOU, KGSHIN}@EECS.UMICH.EDU

Elke A. Rundensteiner
Department of Computer Science
Worcester Polytechnic Institute, Worcester, Massachusetts 01609-2280
RUNDENST@CS.WPI.EDU

ABSTRACT

Scheduling periodic hard real-time tasks is of great importance to many real-time applications, such as open-architecture machine tool controllers and avionics systems. The rate-monotonic scheduling algorithm has been proven to provide an optimal static-priority assignment under idealized conditions. However, some of these conditions are not met in a real computer system. In particular, the release times of tasks can deviate from the specified time instants because of operating system software timer unpredictability. In this paper, we investigate the timer behaviors in three commercial real-time operating systems, VxWorks, QNX and pSOSystem. Based on our findings, we propose an empirical task schedulability model, called RMTU (Rate-Monotonic in the presence of Timing Unpredictability), to augment the rate-monotonic scheduling theory in order to handle timing unpredictability. We then design an approach to systematically derive the model parameters by measurements. With RMTU, task deadlines can be empirically guaranteed. The validity of RMTU is supported by our measurement data. These results are useful not only to control application developers, but also to real-time practitioners at large.

1 Introduction

Scheduling periodic hard real-time tasks is of great importance to many real-time applications [20], such as open-architecture machine tool controllers [16] and avionics systems. For a set of independent, preemptive, and periodic tasks, whose hard deadlines are the same as their respective periods, Liu and Layland [14] proved that the *rate-monotonic* (RM) priority assignment is an optimal static-priority scheduling algorithm. A scheduling algorithm is said to be *static* if the priorities are not changed once they are assigned to the tasks; otherwise, it is *dynamic*.

This research was supported in part by the National Science Foundation under Grants DDM-9313222 and IRI-9309076, and the Engineering Research Center for Reconfigurable Machining Systems.

However, this result was derived under idealized conditions, some of which do not hold in a real computer system. In particular, we observe that the intervals of periodic tasks may vary because the timers that realize the periodicity are not perfect [25]. The precise times of operating system (OS) timer firings can deviate from the nominal time instants, thus introducing variation in actual task periods. This can adversely affect the performance of scheduling algorithms in terms of deadline miss ratios.

To address this problem, we propose an empirical task schedulability model, called RMTU (*Rate-Monotonic in the presence of Timing Unpredictability*), to augment the original RM scheduling algorithm to handle timing unpredictability. In particular, we introduce two parameters to capture the timer delay and the CPU utilization by the OS. These model parameters are determined empirically and systematically for each target system by measuring the execution of a set of predetermined tasks. The model is empirical in the sense that its parameters are derived from measurement data and validated experimentally. We characterize timer behaviors on three commercial real-time operating systems (RTOSs), VxWorks, QNX and pSOSystem, and validate RMTU in a VxWorks testbed.

The remainder of the paper is organized as follows. Sections 2 describes our findings of timer characteristics in three commercial RTOSs—VxWorks, QNX and pSOSystem. In Sections 3 and 4, we describe RMTU and a systematic approach to derive model parameters, respectively. We discuss the results in Section 5. Finally, we present related work and conclusions in Sections 6 and 7, respectively.

2 Software timer characteristics

The RM scheduling theory [14] is very simple and elegant. However, not all its assumptions hold in real-world computer systems. In particular, we notice that, in a multi-tasking environment, periodic tasks rely on the OS timer to realize the periodicity. The timer service typically consists of one or more functions, such as the POSIX [4]

functions *timer_create()*, *timer_delete()*, *timer_gettime()* and *timer_settime()*. A task uses a software timer to generate an event periodically. After each invocation of the task is completed, the task goes into sleep until the timer wakes it up with the next timer event.

Clearly, the characteristics of the OS timer service have potentially significant impact on the performance of periodic tasks. We now present the experiments to measure the timer behaviors of three commercial RTOSs—VxWorks, QNX and pSOSystem.

2.1 VxWorks timer

Our first set of timer experiments is conducted on a Motorola MVME147 board. This is a VMEbus-based processor board with a Motorola 30 MHz 68030 CPU and 4 MB of RAM. It runs VxWorks (version 5.1.1) [24]. VxWorks is a development and execution environment for real-time and embedded applications on a wide variety of target processors. It includes a high-performance scalable RTOS which executes on a target processor. The system clock resolution of the VxWorks kernel is 1 *ms* (1000 ticks/second).

The VME StopWatch [3] is used for timing measurement. It is a board that is plugged into the VME chassis and time-stamps bus events. These events are reads or writes to specific VME extended addresses. In our experiments, a simple inline function call is used to generate the events. The internal clock resolution of the VME StopWatch is 25 *ns*.

We use the POSIX timer to periodically trigger a simple task τ which consists of two consecutive function calls that generate events *E1* and *E2*, respectively. There is no other code between the function calls. Therefore, the elapsed time between *E1* and *E2* is the execution time of one function call, while the elapsed time between two consecutive *E1* (or *E2*) events is the task interval (or timer interval, i.e., the interval between two consecutive timer firings). Table I lists the statistics of the task τ with a 10 *ms* period using the POSIX timer, while Figures 1 and 2 show the histograms of the measured timer intervals and event-generating function execution times, respectively.

	Execution Time	Interval
Sample Size	963	962
Mean (µs)	1.90	9999.9
Standard Deviation (µs)	0.0383	202.7
Min (µs)	1.85	7716.3
Max (µs)	2.20	11964.9

Table I. VxWorks POSIX timer statistics.

From the measurements, we have two important observations about timer characteristics. First, timer intervals vary around the nominal period (see Fig. 1). This is because there are other OS activities besides the user task that consume CPU cycles. Many OS tasks run at higher priorities than user tasks. For example, the VxWorks process manager *tExcTask* runs at the highest priority. The hardware interrupts generated by devices (e.g., network devices) are also serviced at higher priorities by the OS kernel. All these inevitably cause fluctuations in the timer firings as well as task execution times (see Fig. 2).

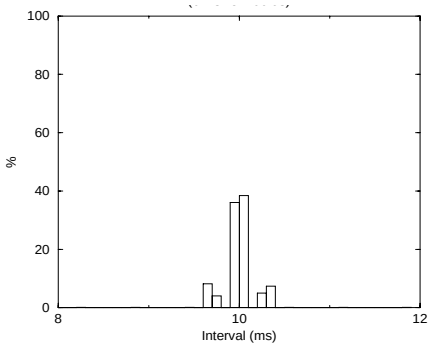

Figure 1. Histogram of VxWorks timer intervals (bin width: 100 µs)

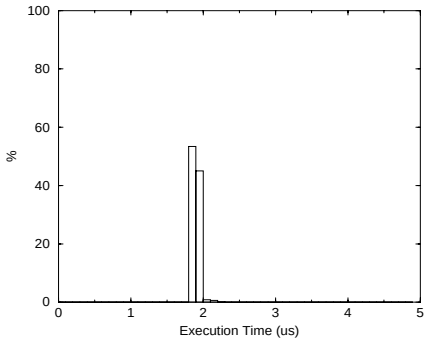

Figure 2. Histogram of VxWorks event-generating function execution times (bin width: 0.1 µs)

Second, timers have "memory" behavior. By examining the actual measurement data (not shown here) [27], we observe that whenever an interval deviates from the nominal period by at least one or two percentage points, the next interval almost always swings in the opposite direction by roughly the same amount. For example, the longest interval of 11,964.9 µs in Table I is followed immediately by the shortest interval of 7,716.3 µs, which is in turn followed by a slightly longer interval of 10,316.9 µs. This can be explained by the use of absolute time inside the OS kernel. What happens is that one timer firing is late (hence the longer interval) because of other higher-priority system activities. The RTOS still tries to fire the timer at its next originally scheduled time. Therefore, the interval immediately following the longer one is shorter than the nominal period. In this example, the second timer firing is slightly early, which further shortens the second interval. The third firing is on time. The average of these three intervals is 9,999.4 µs, which is very close to the nominal period of 10 *ms*.

The nature of these two timer characteristics, variation and "memory" behavior, suggests that they are generic phenomena of RTOSs. To validate this, we con-

duct similar timer experiments using two other RTOSs—QNX and pSOSystem.

2.2 QNX timer

QNX is a commercial, micro-kernel, POSIX-compliant RTOS [17]. It uses a priority-based, preemptive kernel scheduler. Our experiments are conducted on a XYCOM XVME-674/16 board running QNX version 4.22. This board is a VMEbus PC/AT processor module with an Intel 66 MHz 80486DX2, 32 Mbytes dual-access DRAM, SVGA and IDE controllers. The QNX system clock resolution is set to 50 µs and the VME StopWatch is used for the timing measurement.

For the simple task τ with a 10 *ms* period (same as that in the VxWorks experiments) using POSIX timer functions, Table II gives measured task execution time and interval statistics. The QNX POSIX timer also exhibits interval variation. The timer "memory" behavior is present as well, which can be illustrated by the following sequence of six consecutive timer intervals extracted from the measurement data: 9974.3, 9976.9, 10257.5, 9782.0, 9966.4, and 10005.0 µs. The first two intervals are close to the nominal period. The third interval is abnormally long, which is compensated by the next abnormally short interval. The fifth and sixth intervals are again close to the nominal period.

	Execution Time	Interval
Sample Size	1950	1949
Mean (µs)	4.27	9996.3
Standard Deviation (µs)	4.82	30.9
Min (µs)	1.45	9739.1
Max (µs)	16.00	10257.6

Table II. QNX POSIX timer statistics.

2.3 pSOSystem timer

The pSOSystem is another commercial modular high-performance RTOS designed specifically for embedded microprocessors [5]. It includes a real-time multi-tasking kernel pSOS$^+$. Though the pSOSystem (version 1.1) is not POSIX-compliant, it has system calls similar to POSIX timer functions. We substitute the POSIX timer functions with pSOSystem-specific functions in our experiments.

We use a setup similar to that for VxWorks and QNX, where pSOS$^+$ runs on a VMEbus-based processor board—Ironics IV3207 (Motorola 25 MHz 68040 with 4 MB RAM). The system clock resolution of pSOS$^+$ is 10 *ms* (100 ticks/second). Again, timer interval variation (Table III) and "memory" behavior are observed in the measurement data.

2.4 Timer delay factor

Suppose there is a set of *m* independent preemptive periodic tasks, τ_1, τ_2, …, τ_m, with periods T_1, T_2, …, T_m, deadlines D_1, D_2, …, D_m, and worst-case nominal execution times C_1, C_2, …, C_m, and initial release times I_1, I_2, …, I_m, respectively. For each task τ_i, *i=1,2,…,m*, its deadline is assumed to be equal to its period, i.e., $D_i=T_i$. Without loss of generality, the task periods are assumed to be sorted in a non-decreasing order: $T_1 \le T_2 \le \ldots \le T_m$.

	Execution Time	Interval
Sample Size	999	998
Mean (µs)	0.68	10000.2
Standard Deviation (µs)	0.011	82.5
Min (µs)	0.65	9764.2
Max (µs)	0.78	10244.8

Table III. pSOSystem timer statistics.

Figure 3 illustrates the effect of the timer "memory" behavior on task schedulability. The initial timer firing is on time at time *I*, the second is late by *v* time units at *I+T+v*, and the third is again on time at *I+2T*. The net effect is a longer interval of *T+v*, followed by a shorter interval of *T-v*. We call the parameter *v timer delay factor*. If *v > T-C*, the job in that period will miss its deadline.

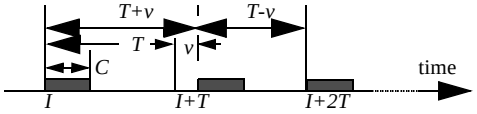

Figure 3. Task execution with timer variation.

Considering that the timer delay is caused by higher-priority OS activities, it should be independent of nominal timer intervals. To verify this, we measure the variation of timers with different nominal intervals. We use the VxWorks setup to run 8 periodic tasks, one at a time, and measure their periods. The nominal periods of these tasks range from 5 ms to 200 ms. The typical range of control task periods is from 1 ms to 100 ms. Since the tick size is 1 ms, it is inappropriate to run tasks with a period that is very close to, or less than, 1 ms.

Table IV summarizes the measurement results. The mean, minimum and maximum periods in the table are normalized by their respective nominal task periods so that it is easier to compare the measurements of tasks with different periods. For example, for the 10 ms task (the second row in the table), the actual mean, minimum and maximum intervals are 10000.38, 8391 and 11487 µs, respectively. We can see that the worst-case timer deviations are between one and two milliseconds and have no linear relationship with nominal timer periods.

Task Period (*ms*)	Sample Size	Interval Mean (µs)	Standard Deviation (µs)	Interval Min (µs)	Interval Max (µs)
5	1213	1.26	132	-1669	1544
10	802	0.38	175	-1609	1487
20	607	-0.33	196	-1653	1385
30	539	-0.71	247	-1373	1393
50	394	0.42	254	-1060	1156
70	522	1.40	224	-1185	1264
100	417	3.54	150	-1425	1608
200	510	1.06	133	-1565	1540

Table IV. Statistics of timers with different nominal periods.

3 RMTU: an empirical schedulability model

As described in the previous section, we have observed the timer variation and "memory" behavior in three popular commercial RTOSs. The nature of such timing unpredictability leads us to believe that they are generic phenomena. In [25] and [26], we further show that timing unpredictability has an adverse impact on scheduling algorithm performance in terms of deadline miss ratios. For example, some task deadline guarantees that would have been provided by RM under idealized conditions may no longer be possible due to timing unpredictability. Therefore, it is important to take timing unpredictability into account when checking the schedulability of tasks. Specifically, we propose an empirical task schedulability model, called RMTU (*Rate-Monotonic in the presence of Timing Unpredictability*), which extends the original RM scheduling theory to handle timing unpredictability. By *empirical*, we mean that the model parameters reflect some specific properties of the target system and thus are experimentally derived from the measurement data of the target system.

RMTU quantifies the effect of timing unpredictability on task schedulability by extending the RM scheduling theory with the following sufficient condition:

$$\forall \tau_i, 1 \le i \le m, U_s + \frac{C_1}{T_1} + \ldots + \frac{C_i}{T_i} + \frac{v}{T_i} \le i\left(2^{\frac{1}{i}} - 1\right) \quad \text{(EQ 1)}$$

where U_s is a constant representing the CPU utilization of OS activities, such as the OS kernel scheduling and interrupt handling; and v is the worse-case timer delay factor, which is a constant, independent of task periods T_i. RMTU uses U_s and v to capture the unpredictability of RTOS. These two parameters can be determined empirically as described in the next section.

Once we obtain the parameters U_s and v and know the task characteristics C_i and T_i, we can use EQ 1 to determine if all the hard deadlines can be met. This model results in a tighter upper bound of CPU utilization for RM task scheduling in the presence of RTOS unpredictability than the original RM.

4 Derivation of model parameters

Now that we have introduced the empirical task schedulability model RMTU, we will describe a systematic approach to deriving model parameters.

4.1 Assumptions and derivation

To apply RMTU, we need to know all the parameters in EQ 1. T_1, T_2, …, T_m are the task periods, which are realized using OS timers in applications. In an actual implementation of tasks, the periods could differ from the specified values. Since timers can deliver mean periods very close (typically, within 0.1%) to their respective task periods (see also the timer measurement data in Section 2), we will use the task periods as given in our schedulability analysis.

Theoretically, the worst-case nominal task execution times C_1, C_2, …, C_m may be obtained by analyzing the task code, assuming that the task execution times are bounded. In a real system, however, the same OS activities that cause timer glitches can affect task execution times as well. Any CPU time used by OS activities during the execution of a user task increases the task response time. Since this increase is not caused by other user tasks, it effectively increases the task's execution time. Therefore, the *measured* worst-case execution times reflect more accurately the actual system performance than those obtained from code analysis and should be used in determining task schedulability. Finally, the worst-case timer delay factor v can only be obtained experimentally.

Our systematic approach to parameter derivation includes experiments using a single task, in which case RMTU can be written as EQ 2. For simplicity, the task subscript and the less-than sign are dropped. This simplification is acceptable because the derived model parameters will be plugged back into EQ 1.

$$C = (1 - U_s)T - v \quad \text{(EQ 2)}$$

We define the *available* CPU utilization (U_{avail}) as the percentage of CPU cycles that is available to user tasks, which is equal to $1\text{-}U_s$. We further define the *achievable* CPU utilization (U_{achiev}) as the summation of the worst-case execution time to period ratios of individual tasks, which implies the *maximal* CPU utilization by user tasks without missing any deadlines. U_{achiev} has the same value as the CPU utilization in the original RM theory.

In EQ 2, the task period T is given, while the worst-case execution time C can be measured. The available CPU utilization $1\text{-}U_s$ and the timer delay factor v can be derived by using the approach we will describe next.

4.2 Systematic approach to parameter derivation

Based on EQ 2, we design the following systematic approach to determining the model parameters empirically:

1. Run a task with a fixed period and an adjustable execution time at the highest priority allowed by the RTOS.
2. Measure the start and completion time of each task invocation to check if it misses its deadline. The number of measured task invocations should be as large as possible and no less than a few hundreds for a given task execution time.[1]
3. If the task misses its deadline, decrease the task execution time and repeat Step 2; otherwise, increase it. Adjust the task execution time until the task does not miss its deadline *and* a small[2] increase in its execution time would cause it to miss its deadline.
4. Record the worst-case execution time[3] C and the

1. Density estimation techniques may be used to determine the number of measurement and confidence level of the results [21].
2. In our experiments, a small change is typically less than 1% of total task execution time.

corresponding task period *T*.

5. Repeat Steps 1-4 for several (e.g., five or more) different task periods in the range of the target application task periods.[4]
6. After obtaining (*C*, *T*) value pairs from Steps 1-5, use linear regression to derive the available CPU utilization ($1\text{-}U_s$) and the timer delay factor *v*.

4.3 Experimental results

We conducted the experiments described above on the computer system running VxWorks described in Section 2.1. Fig. 4 shows the relationship between the *measured* achievable CPU utilization and the task period. While the RM scheduling theory predicts that the *achievable* CPU utilization should be 1 regardless of different periods in the single-task case, Fig. 4 clearly shows an upward trend in achievable CPU utilization as the task period increases (as indicated by EQ 2). This is because of the period-independent timer delay factor in RMTU.

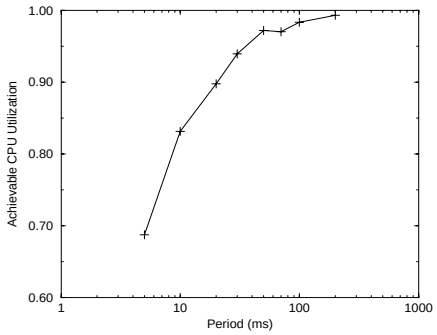

Figure 4. Single-task achievable CPU utilization versus task period.

Table V lists the measurement and computation results. The first eight rows of the table show the data for the tasks whose periods range from 5 to 200 ms. The last three rows are the results of linear regression. Using the measured maximum execution times, linear regression produces a timer delay factor *v* of 1.802 ms and an available CPU utilization ($1\text{-}U_s$) of 1.0016 (fourth column in the table). We will discuss why U_{avail} could be greater than 1 in Section 5. The correlation coefficient is very close to 1, indicating a strong positive correlation between the task period and measured worst-case execution time.

4.4 Model validation

To validate the empirical model, we have run several sets of three and five independent tasks in the same VxWorks setup where the model parameters were derived. Tasks are scheduled using the RM priority assignment. RMTU provides tighter upper bound on CPU utilization than RM. We show that no tasks missed their deadlines when the CPU utilization is less than, or equal to, the schedulability threshold given by RMTU.[5] This indicates that the model is effective as a sufficient condition for task schedulability with the RM priority assignment in the presence of timing unpredictability in these experiments.

Task Period *T* (*ms*)	Sample Size	Achiev. CPU Utilization	Max Execution Time *C* (*ms*)	Mean Execution Time (*ms*)	Min Execution Time (*ms*)
5	1214	0.688	3.438	2.771	2.695
10	803	0.831	8.314	7.726	7.650
20	608	0.898	17.954	17.672	17.588
30	540	0.939	28.184	27.700	27.583
50	395	0.972	48.597	48.002	47.962
70	523	0.970	67.920	67.395	67.259
100	418	0.983	98.347	97.694	97.651
200	511	0.993	198.606	197.698	197.586
timer delay factor *v* (*ms*)			1.802	2.271	2.350
available CPU utilization $1\text{-}U_s$ (*ms*)			1.0016	0.9996	0.9995
regression correlation coefficient			+1.00000	+1.00000	+1.00000

Table V. Single task experiment results.

5 Discussion

There are a few important observations from our experimental measurements in Section 4. First, RMTU gives a tighter upper bound of the achievable CPU utilization than the original RM scheduling theory. Second, the tasks can typically reach better achievable CPU utilization than predicted by either RM or RMTU, because most task executions take less time than the worst case.

Third, the available CPU utilization ($1\text{-}U_s$) could be greater than one. At the first glance, it appears inappropriate that the available CPU utilization can be greater than one. However, the available CPU utilization is not the actual CPU utilization. When the worst-case task execution times are used in EQ 2 to derive model parameters, the available CPU utilization can be over-estimated (thus resulting in a value larger than one). This is because only a very small number of task invocations actually take that much time to complete.

RMTU is premised on that the system characteristics revealed by single-task experiments are applicable to multiple tasks. However, we found that the presence of other tasks does have an effect on the variation of task execution times, even if the task has the highest priority [27]. It was also observed that a single application can receive periodic callbacks or timer notifications every 1 ms, for instance, with a high degree of precision. But when two or more applications try to do this simultaneously, there are outliers lasting tens of milliseconds that never appear in the single application case. This happens even though the system is essentially completely idle [7]. While RMTU (or

3. This is actually the worst-case response time, i.e., the elapsed time between the start and the finish of the task. With our approach, the effect of OS activities on task execution time is also factored into our empirical schedulability model.

4. This is intended to improve the precision of linear regression.

5. Measurement results [27] are omitted due to space limit.

any empirical models) cannot theoretically provide hard guarantees, it works well for practical purposes as shown by our experiments. For applications that require deadline guarantees but not necessarily hard, we instead propose probabilistic deadline guarantees [27].

6 Related work

RM has been extended for the cases with $D_i \le T_i$ [8, 10, 11, 12], with deadlines being an integral multiple of periods [11, 13], and with $D_i > T_i$ [19]. Rajkumar *et al*. [18] and Lortz and Shin [15] studied scheduling issues for periodic tasks with blocking due to non-preemptive critical task sections or mutually exclusive access to shared resources. All of the above extensions of the RM scheduling algorithm assume ideal timers.

Kettler, Katcher and Stronsnider [9] proposed an engineering methodology that allows users to accurately model and evaluate RTOSs, thus providing a framework to account for implementation costs in real-time scheduling theory. Their approach requires a high degree of expertise to create a valid model for any given RTOS. Such expertise may be scarce among users. Furthermore, as RTOSs become more sophisticated and new releases come out once or twice a year, modeling them becomes more difficult. Instead, we design a set of systematic experiments to characterize the underlying RTOS and provide a simple empirical sufficient condition for applying the RM scheduling theory to practical systems.

Audsley *et al*. [1, 22] extended static-priority scheduling techniques to address timer jitter. Their solution is to use a non-preemptive approach trying to avoid or at least minimize the problem. Baruah *et al*. [2] provided a formal feasibility analysis of dynamic-priority systems exhibiting jitter. However, we observe that the overhead of run-time priority change can be prohibitively expensive (e.g., QNX's *setprio()* system call takes more than 100 µs). A custom-built scheduling system is not always a viable alternative either. We instead address the problem by incorporating jitter effects in static-priority systems using commercial RTOSs.

Jeffay and Stone [6] studied the feasibility and schedulability problems for periodic tasks in the presence of interrupt handlers. While their solutions are computationally feasible, some of their assumptions are overly-simplistic. For example, they assume that interrupt handlers are strictly periodic, which is rarely the case in practice.

7 Conclusions

Many scheduling algorithms, e.g., rate-monotonic, may not work well in practice because some of their idealized assumptions do not hold in a real computer system. In particular, we observed timer period variations in three commercial RTOSs—VxWorks, QNX and pSOSystem. In order to handle such timing unpredictability, we proposed an empirical model—RMTU—to extend rate-monotonic. We further designed a set of systematic experiments, from which the model parameters can be empirically derived. Our experiments verified the validity of RMTU.

References

[1] N.C. Audsley, I.J. Bate, and A. Burns, "Putting Fixed Priority Scheduling Theory into Engineering Practice for Safety Critical Applications," *RTAS'96*, pp. 2-10, 1996.

[2] S. Baruah, D. Chen, and A. Mok, "Jitter Concerns in Periodic Task Systems," *RTSS'97*, pp. 68-77, 1997.

[3] S. Han, K.G. Shin, and J. Park, "A Non-intrusive Distributed Monitoring Support in Fault Injection Experiments," *4th IEEE International Workshop on Evaluation Techniques for Dependable Systems*, 1995.

[4] IEEE, *IEEE Guide to POSIX Open System Environment*, 1995.

[5] Integrated Systems Inc., *pSOSystem/68K User's Manual*, 1992.

[6] K. Jeffay, and D.L. Stone, "Accounting for Interrupt Handling Costs in Dynamic Priority Task Systems," *RTSS'93*, pp. 212-221, 1993.

[7] M. Jones, private communications, 1998.

[8] M. Joseph, and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, Vol. 29, No. 5, pp. 390-395, 1986.

[9] K.A. Kettler, D.I. Katcher, and J.K. Strosnider, "A Modeling Methodology for Real-Time/Multimedia Operating Systems," *RTAS'95*, pp. 15-26, 1995.

[10] M.H. Klein, *A Practitioner's handbook for real-time analysis guide to rate monotonic analysis for real-time systems*, Kluwer, 1993.

[11] M.H. Klein, J.P. Lehoczky, and R. Rajkumar, "Rate-Monotonic Analysis for Real-Time Industrial Computing," *IEEE Computer*, January 1994, pp. 24-32.

[12] J.P. Lehoczky, et al., "Fixed Priority Scheduling Theory for Hard Real-Time Systems," A.M. van Tilborg and G.M. Koob, eds., *Foundations of Real-Time Computing: Scheduling and Resource Management*, Kluwer Academic Publishers, pp. 1-30, 1991.

[13] J.P. Lehoczky, "Real-Time Resource Management Techniques," J.J. Marciniak, ed., *Encyclopedia of Software Engineering*, John Wiley and Sons, pp. 1011-1020, 1994.

[14] C.L. Liu, and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, Vol. 20, No. 1, January 1973, pp. 46-61.

[15] V.B. Lortz, and K.G. Shin, "Semaphore Queue Priority Assignment for Real-Time Multiprocessor Synchronization," *IEEE Trans. on Software Engineering*, Vol. 21, No. 10, October 1995, pp. 834-844.

[16] Open-Architecture Controls Team, *Developer's Guide for Open-Architecture Control of the Robotool*, Dept. of Electrical Engineering & Computer Science and Dept. of Mechanical Engineering & Applied Mechanics, U. of Michigan, November 1995.

[17] QNX Software Systems Ltd., *QNX 4 Operating System: System Architecture*, 1993.

[18] R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-time synchronization protocols for multiprocessors," *RTSS'98*, pp. 259-269, 1998.

[19] W.K. Shih, J.W.S. Liu, and C.L. Liu, "Modified Rate-Monotonic Algorithm for Scheduling Periodic Jobs with Deferred Deadlines," *IEEE Transactions on Software Engineering*, Vol. 19, No. 12, December 1993.

[20] K.G. Shin, and P. Ramanathan, "Real-Time Computing: A New Discipline of Computer Science and Engineering," *IEEE Proceedings*, Vol. 82, No. 1, Jan. 1994, pp. 6-24.

[21] B.W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, 1986.

[22] K.W. Tindell, A. Burns and A.J. Wellings, "An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks," *Real-Time Systems*, 6, 1994, pp. 133-151.

[23] S. Vestal, "On the Accuracy of Predicting Rate Monotonic Scheduling Performance," *Tri-Ada*, December 1990.

[24] Wind River Systems, *VxWorks Reference Manual 5.1*, 1993.

[25] L. Zhou, K.G. Shin, E.A. Rundensteiner, and N. Soparkar, "Probabilistic Real-Time Data Access with Interval Constraints," *RTDB'96*, pp. 15-22, 1996.

[26] L. Zhou, M.J. Washburn, K.G. Shin, and E.A. Rundensteiner, "Performance Evaluation of Modular Real-Time Controllers," *IMECE'96*, DSC-Vol. 58, pp. 299-306, 1996.

[27] L. Zhou, "Real-Time Performance Guarantees in Manufacturing Systems," *Ph.D. Dissertation* (draft), U. of Mich., 1998.