A Multiframe Model for Real-Time Tasks *

Aloysius K. Mok, Deji Chen
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
{mok,cdj}@cs.utexas.edu

Abstract

The well-known periodic task model of Liu and Layland [1] assumes a worst-case execution time bound for every task and may be too pessimistic if the worst-case execution time of a task is much longer than the average. In this paper, we give a multiframe real-time task model which allows the execution time of a task to vary from one instance to another by specifying the execution time of a task in terms of a sequence of numbers. We investigate the schedulability problem for this model for the preemptive fixed priority scheduling policy. We show that a significant improvement in the utilization bound can be established in our model.

*This work is supported by a grant from the Office of Naval Research under grant number N00014-94-1-0582.

1 Introduction

The well-known periodic task model by Liu and Layland(L&L) [1] assumes a worst-case execution time bound for every task. While this is a reasonable assumption for process-control-type real-time applications, it may be overly conservative [4] for situations where the average-case execution time of a task is significantly smaller than that of the worst-case. In the case where it is critical to ensure the completion of a task before its deadline, the worst-case execution time is used at the price of excess capacity. Other approaches have been considered to make better use of system resources when there is substantial excess capacity. For example, many algorithms have been developed to schedule best-effort tasks for resources unused by hard-real-time periodic tasks; aperiodic task scheduling has been studied extensively and different aperiodic server algorithms have been developed to schedule them together with periodic tasks [6, 7, 8, 9]. In [10], etc., the *imprecise computation* model is used when a system cannot schedule all the desired computation. We have also investigated an adaptive scheduling model where the timing parameters of a real-time task may be parameterized [3]. However, none of the work mentioned above addresses the scheduleability of real-time tasks when the execution time of a task may vary greatly but follows a known pattern. In this paper, we propose a *multiframe task model* which takes into account such execution time patterns; we shall show that better schedulability bounds can be obtained.

In the multiframe model, the execution time of a task is specified by a finite list of numbers. By repeating this list, a periodic sequence of numbers is generated such that the execution time of each instance (frame or job) of the task is bounded above by the corresponding number in the periodic sequence. Consider the following example. Suppose a computer system is used to track vehicles by registering the status of every vehicle every 3 time units. To get the complete picture, the computer takes 3 time units to perform the tracking execution, i.e., the computer is 100% utilized. Suppose in addition, the computer is required to execute some routine task which takes 1 time unit and the task is to be executed every 5 time units. Obviously, the computer cannot handle both tasks.

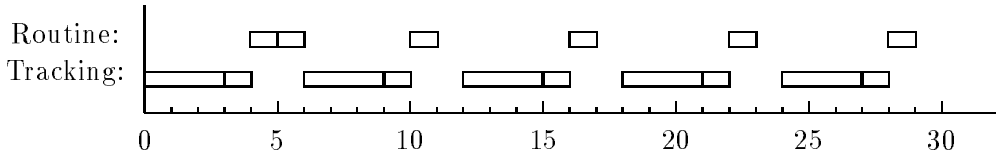

Figure 1: Schedule of the Vehicle Tracking System

However, if the tracking task can be relaxed so that it requires only 1 time unit to execute every other period, then the computer should be able to perform both the tracking and routine tasks (see the timing diagram in figure 1).

This solution cannot be obtained by the L&L model since the worst-case execution time of the tracking task is 3, so that the periodic task set in the L&L model is given by $\{(3,3),(1,5)\}$ (the first component in a pair is the execution time and the second the period). This task set has utilization factor of 1.2 and is thus unscheduleable. Also notice that we cannot replace the tracking task by a pair of periodic tasks $\{(3,6),(1,6)\}$ since a scheduler may defer the execution of the $(3,6)$ task so that its first execution extends past the interval [0,3], while in fact it must be finished by time=3.

In this paper, we shall investigate the schedulability of tasks for our multiframe task model under the preemptive fixed priority scheduling policy. For generality, we allow tasks to be *sporadic* instead of periodic. A sporadic task is one whose requests may not occur periodically but there is a minimum separation between any two successive requests from the same task. A periodic task is the limiting case of a sporadic task whose requests arrive at the maximum allowable rate. We shall establish the utilization bounds for our model which will be shown to subsume the L&L result [1]. To obtain these results, however, we require the execution times of the multiframe tasks to satisfy a fairly liberal constraint. It will be seen that the schedulability bounds can be improved substantially if there is a large variance between the peak and non-peak execution time of a task. Using the multiframe model, we can safely admit more real-time tasks than the L&L model.

The paper is organized as follows. Section 2 presents our multiframe real-time task model,

defines some terminology, and prove some basic results about scheduling multiframe tasks. Section 3 investigates the schedulability bound of the fixed priority scheduler for the multiframe model. Section 4 is the conclusion.

2 The Multiframe Task Model

For the rest of the paper, we shall assume that time values have the domain the set of non-negative real numbers. All timing parameters in the following definitions are non-negative real numbers. We remark that all our results will still hold if the domain of time is the non-negative integers.

Definition 1 *A multiframe real-time task is a tuple* (Γ, P)*, where* Γ *is an array of* N *execution times* $(C^0, C^2, \ldots, C^{N-1})$ *for some* $N \geq 1$*, and* P *is the minimum separation time, i.e., the ready times of two consecutive frames (requests) must be at least* P *time units apart. The execution time of the ith frame of the task is* $C^{((i-1) \ mod \ N)}$*, where* $1 \leq i$*. The deadline of each frame is* P *after its ready time.*

For example, $T_1 = ((2,1),2)$ is a multiframe task with a minimum separation time of 2. Its execution time alternates between 2 and 1. When the separation between two consecutive ready times is always P and the ready time of the first frame of a task is at time=0, the task reduces to a periodic task.

In the proofs to follow, we shall often associate a multiframe task whose Γ has only one element (i.e., N=1) with a periodic task in the L&L model which has the same execution time and whose period is the same as the minimum separation of the multiframe task. For example, task $T_1 = ((1),5)$ has only one execution time and its corresponding L&L task is $(1,5)$. We shall call a periodic task in the L&L model an L&L task, and whenever there is no confusion, we shall call a multiframe task simply a task.

Consider a task $T = ((C^0, C^1, \ldots, C^{N-1}), P)$ which has more than one execution time. Let $C^m = max_{i=0}^{N-1} C^i$. We shall call C^m the *peak execution time* of task T. We shall call the pair

(C^m, P) the corresponding L&L task of the multiframe task T.

Definition 2

For a set S of n tasks $\{T_1, T_2, \ldots, T_n\}$:

We call $U^m = \Sigma_{i=1}^{n} C_i^m / P_i$, the peak utilization factor of S.

We call $U^v = \Sigma_{i=1}^{n}((\Sigma_{j=0}^{N-1} C_i^j)/(N_i \cdot P_i))$, the maximum average utilization factor of S:

Given a scheduling policy A, we call $\overline{U_A^m}$ the utilization bound of A if for any task set S, S is scheduleable by A whenever $U^m \leq \overline{U_A^m}$,

We note that U^m is also the utilization factor of S's corresponding L&L task set.

Example 1 *Consider the task set $S = \{T_1, T_2\} = \{((3,1),3),((1),5)\}$. Its corresponding L&L task set $S'$is$\{T_1', T_2'\} = \{(3,3),(1,5)\}$. The peak utilization factor of S is $U^m = 1.2$. The maximum average utilization factor of S is $U^v = 0.867$.*

A pessimistic way to analyze the scheduleability of a multiframe task set is to consider the schedulability of its corresponding L&L task set. This, however, may result in rejecting many task sets which actually are scheduleable. For example, the task set in Example 1 will be rejected if we use the L&L model, whereas it is actually scheduleable by a fixed priority scheduler under RMA (Rate Monotonic Assignment), as we shall show later.

Definition 3 *With respect to a scheduling policy A, a task set is said to be fully utilizing the processor if it is scheduleable by A, but increasing the execution time of any frame of any task will result in the modified task set being unscheduleable by A.*

We note that $\overline{U_A^m}$ is the greatest lower bound of all fully utilizing task sets with respect to the scheduling policy A.

Lemma 1 *For any scheduling policy A, $\overline{U_A^m} \leq 1$.*

Proof. We shall prove this by contradiction. Suppose there is an $\overline{U_A^m}$ larger than 1, we arbitrarily form a task set $S = \{(\Gamma_1, P), (\Gamma_2, P), \ldots, (\Gamma_n, P)\}$ with $\Sigma_{i=1}^n C_i^m/P = \overline{U_A^m}$ where C_i^m is the peak execution time of T_i. So we have $\Sigma_{i=1}^n C_i^m > P$. When the peak frames of all the tasks start at the same time, they cannot be all finished within P by any scheduler, which violates the definition of $\overline{U_A^m}$. So $\overline{U_A^m}$ cannot exceed 1. **QED**.

Lemma 2 *Suppose A is a scheduling policy which can be used to schedule both multiframe and L&L task sets. Let the utilization bound of A be $\overline{U_A^m}$ for multiframe task sets. Let the utilization bound of A for the corresponding L&L task sets be $\overline{U_A^c}$. Then $\overline{U_A^m} \geq \overline{U_A^c}$.*

Proof. Proof is by contradiction. Consider a task set S of size n. Suppose $U^m \leq \overline{U_A^c}$ and the set is unscheduleable. Its corresponding L&L task set S' has the same utilization factor as U^m. S' is scheduleable.

Suppose the ith frame of task T_j miss its deadline at time t_j. For every task T_k, $1 \leq k \leq n$, locate the time point t_k which is the ready time of the latest frame of T_k such that $t_k \leq t_j$. We transform the ready time pattern as follows. In the interval from 0 to t_k, we push the ready times of all frames toward t_k so that the separation times of all consecutive frames are all equal to P_k. We now set all execution times to be the peak execution time. If $t_j - t_k > P_k$ for some k, we add more peak frames of T_k at its maximum rate in the interval between t_k and t_j. The transformed ready time pattern is at least as stringent as the original case. So the ith frame of T_j still misses its deadline. However, the transformed case is actually a ready time pattern of S' which should be scheduleable, hence a contradiction **QED**.

Is the inequality in Lemma 2 strict? Intuitively, if U^m of a task set is larger than $\overline{U_A^c}$ and there is not much frame-by-frame variance in the execution times of the tasks in the set, then the task set is unlikely to be scheduleable. However, if the variance is sufficiently big, then the same scheduling policy will admit more tasks. This can be quantified by determining the utilization bound for our task model. We shall show how to establish an exact bound if the execution times of the tasks

satisfy a rather liberal restriction.

Definition 4 *Let C^m be the maximum in an array of execution times $(C^0, C^1, \ldots, C^{N-1})$. This array is said to be AM (Accumulatively Monotonic) if $\Sigma_{k=m}^{m+j} C^{(k \bmod N)} \geq \Sigma_{k=i}^{i+j} C^{(k \bmod N)}$, $1 \leq i \leq N-1$, $1 \leq j < N-1$. A task $T = \{(C^0, C^1, \ldots, C^{N-1}), P)\}$ is said to be AM if its array of execution times is AM.*

Intuitively, an AM task is a task whose total execution time for any sequence of $L \geq 1$ frames is the largest among all size-L frame sequences when the first frame in the sequence is the frame with the peak execution time. For instance, all tasks in Example 1 are AM. We note that tasks in multimedia applications usually satisfy this restriction.

In the following section, we assume that all tasks satisfy the AM property. It will be seen that without loss of generality, we can assume that the first component of the array of execution time of every task is its peak execution time, i.e., C^m=C^0

3 Fixed Priority Scheduling

In this section, we shall show that, for the preemptive fixed-priority scheduling policy, the multiframe task model does have a higher utilization bound than the L&L model if we consider the execution time variance explicitly. The utilization bound for the L&L model is given by the following theorem in the much cited paper [1].

Theorem 1 (Theorem 5 from [1]) *For L&L task sets of size n, the utilization bound of the preemptive fixed priority schuduling policy is $n(2^{1/n} - 1)$.*

Definition 5 *The critical instance of a multiframe task is the period when its peak execution time is requested simultaneously with the peak execution times of all higher priority tasks, and all higher priority tasks request execution at the maximum rate.*

Theorem 2 *For the preemptive fixed priority scheduling policy, a multiframe task is scheduleable if it is scheduleable in its critical instance.*

Proof. Suppose a task $T_k = (\Gamma_k, P_k)$ is scheduleable in its critical instance. We shall prove that all its frames are scheduleable regardless of their ready times.

First, we prove that the first frame of T_k is scheduleable. Let T_k be ready at time t and its first frame finishes at t_{end}. We trace backward in time from time=t to locate a point t_0 when none of the higher priority tasks was being executed. t_0 always exists, since at time 0 no task is scheduled. Let us pretend that T_k's first frame becomes ready at time t_0. It will still finish at time t_{end}. Now let us shift the ready time pattern of each higher priority task such that its frame which becomes ready after t_0 now becomes ready at t_0. This will only postpone the finish time of T_k's first frame to a point no earlier than t_{end}, say t'_{end}. In other words, $T_{end} \leq t'_{end}$. Then for each higher priority task, we shift the ready time of every frame after t_0 toward time=0, so that the separation between two consecutive frames is always the minimum separation time. This will further postpone the finish time of T_k's first frame to no earlier than T'_{end}, say T''_{end}. In other words, $t'_{end} \leq t''_{end}$. Now, we shift all higher priority tasks by by frames until the peak frame starts at t_0. Since Γ_k is AM, this shifting has the effect of postponing the finish time of T_k to t'''_{end}, $t''_{end} \leq t'''_{end}$. By construction, the resulting request pattern is the critical instance for T_k. Since T is scheduleable in its critical instance, we have $t'''_{end} - t_0 \leq P_k$, so $t_{end} - t \leq P_k$, which means T_k's first frame is scheduleable.

Next, we prove that all other frames of T_k are also scheduleable. This is done by induction.

Induction base case: The first frame of T_k is scheduleable.

Induction step: Suppose first i frames of T_k are scheduleable. Let us consider the $(i+1)$th frame and apply the same argument as before. Suppose that this frame starts at time t and finishes at t_{end}. Again, we trace backward from t along the time line until we hit a point t_0 when no higher priority tasks is being executed. t_0 always exists, since no higher priority task is being executed at the finish time of the ith frame. Let the $(i+1)$th frame start at time t_0. It will still finish at time t_{end}. Now shift the requests of each higher priority task such that its frame which starts

after t_0 now starts at t_0. This will only postpone the finish time of T_k's $(i+1)$th frame to a point in time no earlier than t_{end}, say t'_{end} where $t_{end} \leq t'_{end}$. Then for each higher priority task, we shift the ready time of every frame after t_0 toward time=0 so that the separation time between any two consecutive frames is always the minimum separation time of the task. This will further postpone the finish time of T_k's $(i+1)$th frame to no earlier than T'_{end}, say T''_{end}. In other words, $t'_{end} \leq t''_{end}$. Now for all higher priority tasks, we shift them by frames until the peak frames start at t_0. Again, since Γ is AM, this further postpones the finish time of T_k's $(i+1)$th frame to t'''_{end} where $t''_{end} \leq t'''_{end}$. This last case is actually the critical instance for T_k. Since T_k is scheduleable in its critical instance, we have $t'''_{end} - t_0 \leq P_k$, so $t_{end} - t \leq P_k$, which means T_k's $(i+1)$th frame is also scheduleable. We have thus proved the theorem. **QED**.

We shall say that a task passes its critical instance test if it is scheduleable in its critical instance.

Corollary 1 *A task set is scheduleable by a fixed priority scheduler if all its tasks pass the critical instance test.*

From now on, we can assume, without loss of generality that C^0 is the peak execution time of a task without affecting the schedulability of the task set. This is because we can always replace a task T whose peek execution time is not in the first frame by one whose execution time array is obtained by rotating T's array so that the peek execution time is C^0. From the argument in the proof of theorem 2, it is clear that such a task replacement does not affect the result of the critical instance test.

Example 2 *Task set* $\{((2,1),3),((3),7)\}$ *is scheduleable under rate-monotonic assignment.* $U^m = 2/3 + 3/7 = 1.095$, $U^v = 0.929$. *Both tasks pass their critical instance test. However, its corresponding L&L task set* $\{(2,3),(3,7)\}$ *is unscheduleable under any fixed priority assignment.*

Example 3 *The L&L task set* $\{(3,3),(1,5)\}$ *with utilization factor* 1.2 *is obviously unscheduleable by any scheduling policy. However, if the requirement of the first task is relaxed such that every*

other frame needs only 1 time unit, the task set becomes scheduleable by RMA. This is because the relaxed case is given by the multiframe task set $\{((3,1),3),((1),5)\}$ *which passes the critical instance test.*

We remark that the Example 3 above specifies the vehicle tracking system mentioned at the beginning of this paper. From the above argument, we can now establish its schedulability. These examples also show that even if the total peak utilization exceeds 1, a task set may still be schedulable. Of course, the average utilization must not be larger than 1 for scheduleability.

The complexity of the scheduleability test based on Corollary 1 is $O(P)$, where P is the biggest period.

Theorem 3 *If a feasible priority assignment exists for some multiframe task set, the rate-monotonic priority assignment is feasible for that task set.*

Proof. Suppose a feasible priority assignment exists for a task set. Let T_i and T_j be two tasks of adjacent priority in such an assignment with T_i being the higher priority one. Suppose that $P_i > P_j$. Let us interchange the priorities of T_i and T_j. It is not difficult to see that the resultant priority assignment is still feasible by checking the critical instances. The rate-monotonic priority assignment can be obtained from any priority ordering by a finite sequence of pairwise priority reordering as above. **QED**.

To compute the utilization bound, we need the following lemma.

Definition 6 *Let* $\Psi(n,\alpha)$ *denote the minimum of the expression* $\Sigma_{i=1}^{n-1}(P_{i+1}-P_i)/P_i + (\alpha \cdot P_1 - P_n)/P_n$, *subject to the constraint:* $P_1 \leq ... \leq P_n \leq \alpha \cdot P_1$ *and* $1 < \alpha \leq 2$.

Lemma 3 $\Psi(n,\alpha) = n \cdot (\alpha^{1/n} - 1)$.

Proof. With the substitution $x_i = \log_2 \frac{P_{i+1}}{P_i}$ where $1 \leq i < n$; $x_n = \log_2 \frac{\alpha P_1}{P_n}$, we can compute $\Psi(n,\alpha)$ by:

minimize $\Sigma_{i=1}^{n}(2^{x_i}-1)$ subject to $x_i \geq 0$ and $\Sigma_{i=1}^{n} x_i = \log_2 \alpha$.

This is a strictly convex problem. There is a unique critical point which is the absolute minimum. The symmetry of the minimization problem in its variables means that all x_i's are equal in the solution. So we have $x_i = (\log_2 \alpha)/n$. So $\Psi(n,\alpha) = \Sigma_{i=1}^{n}(2^{x_i}-1) = \Sigma_{i=1}^{n}(2^{(\log_2 \alpha)/n}-1) = n \cdot (\alpha^{1/n}-1)$. **QED**.

Definition 7 *A task set is said to be extremely utilizing the processor if it is scheduleable but increasing peak execution time of the lowest priority task by any amount will result in a task set which is unscheduleable.*

We shall use $\overline{U^e}$ to denote the greatest lower bound of all extremely utilizing task sets.

It is important to note the distinction between fully utilizing and extremely utilizing task sets. It is crucial to the proof of Lemma 4 and Lemma 5.

Lemma 4 *Consider all task sets of size n satisfying the restriction $P_1 < P_2 < ... < P_n < 2 * P_1$. Let $r = min_{i=1}^{n}(C_i^0/C_i^1)$. Then $\overline{U^e} = r \cdot n \cdot ((\frac{r+1}{r})^{1/n} - 1)$.*

Proof. From Theorem 2 and Theorem 3, we only need to consider the case where all tasks start at time 0 and request at their maximum rates thereafter. We can use rate-monotonic priority assignment and check for scheduleability in the interval from time 0 to P_n. Since $P_1 < P_2 < ... < P_n < 2 * P_1$, we know that only C^0 and C^1 are involved in all the critical instance tests.

First, the utilization bound corresponds to the case where every C^0/C^1 equals r, since we can increase C^1 without changing U^m. And increasing C^1 will only take more CPU time. So in the following proof we assume that all the ratios are equal to r.

For any scheduleable and extremely utilizing task set S with $U^m = \overline{U^e}$, we shall prove four claims.

Claim 1: The second request of T_i, $1 \leq i < n$ must be finished before P_n.

Suppose δ of C_i^1 is scheduled after P_n, we can derive a new task set S' by only changing the following execution times of T_i and T_n,

$$\begin{aligned} C_i'^0 &= C_i^0 - \delta \cdot r \\ C_i'^1 &= C_i^1 - \delta \\ C_n'^0 &= C_n^0 + \delta \cdot r \\ C_n'^1 &= C_n^1 + \delta \end{aligned}$$

and arbitrarily reducing other execution times of T_i to maintain the AM property of the execution time arrays. It is easy to show that S' is schedulable and also extremely utilizes the processor.

$$\begin{aligned} U'^m &= U^m + (C_i'^0 - C_i^0)/P_i + (C_n'^0 - C_n^0)/P_n \\ &= U^m + \delta \cdot r \cdot (1/P_n - 1/P_i) \\ &< U^m \\ &= \overline{U^e} \end{aligned}$$

This contradicts the assumption that $\overline{U^e}$ is the minimum of all extremely utilizing task set. So the second request of any T_i $1 \leq i < n$ should be completed before P_n.

Claim 2: If $P_i < (\frac{r}{r+1})P_n$, then $C_i^0 = 0$

If $C_i^m \neq 0$, we can derive a new task set S' by only changing the following execution times of T_i and T_n, and arbitrarily reducing other execution times of T_i to maintain the AM property of the execution time arrays.

$$\begin{aligned} C_i'^0 &= 0 \\ C_i'^1 &= 0 \\ C_n'^0 &= C_n^0 + C_i^1 \cdot (r+1) \\ C_n'^1 &= C_n^1 + C_i^1 \cdot (r+1)/r \end{aligned}$$

It is easy to check that S' is scheduleable and also extremely utilizes the processor.

$$\begin{aligned} U'^m &= U^m + (C_i'^0 - C_i^0)/P_i + (C_n'^0 - C_n^0)/P_n \\ &= U^m + (P_i - (\frac{r}{r+1}))P_n) \cdot C_i^1/((r+1) \cdot P_i \cdot P_n) \\ &< U^m \\ &= \overline{U^e} \end{aligned}$$

This contradicts the assumption that $\overline{U^e}$ is the minimum. So $C_i^0 = 0$.

Claim 3: If $P_i > (\frac{r}{r+1})P_n$, then C_n^0 should be finished before P_i.

Instead of proving claim 3, we prove the following equivalent claim:

Consider an extreme utilizing task set S satisfying claim 1 and claim 2. If the last part of C_n^0 finishes between P_i and P_{i+1}, and $P_i > (\frac{r}{r+1})P_n$, then S does not correspond to the minimal case.

As in claim 2, we can derive a new task set S' by only changing the following execution times of T_i and T_n, and arbitrarily reducing other execution times of T_n to maintain the AM property of the execution time arrays.

$$\begin{aligned} C_i'^0 &= C_i^0 + r \cdot \delta/(r+1) \\ C_i'^1 &= C_i^1 + \delta/(r+1) \\ C_n'^0 &= C_n^0 - \delta \\ C_n'^1 &= C_n^1 - \delta/r \end{aligned}$$

Suppose P_j is the smallest value satisfying $P_j < (\frac{r}{r+1})P_n$. $P_j \leq P_i$. According to claim 1 and claim 2, the second requests of all tasks other than T_n are scheduled between P_j and P_n. Since $P_n - P_j < P_j$, we know the first requests of all tasks other than T_n are all scheduled before P_j. Since S extremely utilizes the CPU, we know that the part of C_n scheduled before P_j is larger than that scheduled after P_j. This guarantees that the new task set S' is still scheduleable and extremely utilizes the CPU.

$$\begin{aligned} U'^m &= U^m + (C_i'^0 - C_i^0)/P_i + (C_n'^0 - C_n^0)/P_n \\ &= U^m + \delta((\frac{r}{r+1})P_n - P_i)/(P_i \cdot P_n) \\ &< U^m \\ &= \overline{U^e} \end{aligned}$$

Hence, the task set S cannot be the minimal case. This establishes claim 3.

Claim 4: If $P_i > (\frac{r}{r+1})P_n$, then the second request of T_i, $i < n$ should be completed exactly at time P_{i+1}.

If the second request of $T_i, 1 \leq i < n$ completes ahead of P_{i+1}, the processor will idle between its completion time and P_{i+1}, which shows S does not extremely utilize processor. So this cannot be true.

If δ of the second request of $T_i, i < n$ completes after P_{i+1}, we can derive a new task set S' by only changing the following execution times of T_i and T_{i+1}, and arbitrarily reducing other execution times of T_i to maintain the AM property of the execution time arrays.

$$\begin{aligned} C_i'^0 &= C_i^0 - r \cdot \delta \\ C_i'^1 &= C_i^1 - \delta \\ C_{i+1}'^0 &= C_{i+1}^0 + r \cdot \delta \\ C_{i+1}'^1 &= C_{i+1}^1 + \delta \end{aligned}$$

Again it is easy to check that S' is schedulable and also extremely utilizes the process.

$$\begin{aligned} U'^m &= U^m + (C_i'^0 - C_i^0)/P_i + (C_{i+1}'^0 - C_{i+1}^0)/P_{i+1} \\ &= U^m + r \cdot \delta \cdot (1/P_{i+1} - 1/P_i) \\ &< U^m \\ &= \overline{U^e} \end{aligned}$$

This contradicts the assumption that $\overline{U^e}$ is the minimum.

So the second request of $T_i, i < n$ should be completed exactly at time T_{i+1}.

From these four claims and Lemma 3, we can conclude:

$\overline{U^e} = min_{k=1}^{n}(r \cdot \Psi(k, \frac{r+1}{r})) = r \cdot \Psi(n, \frac{r+1}{r}) = r \cdot n \cdot ((\frac{r+1}{r})^{1/n} - 1)$. **QED**

Lemma 5 *Let $r = min_{i=1}^{n}(C_i^0/C_i^1)$. For task sets of size n, $\overline{U^e} = r \cdot n \cdot ((\frac{r+1}{r})^{1/n} - 1)$.*

Proof. Again, we assume all C^0/C^1 equals r, and all tasks request at the maximum rate. For any task T_i in an extremely utilizing task set with $P_i * 2 < P_n$, let $P_n = p_i \cdot P_i + q_i$, $p_i > 1$ and $q_i \geq 0$. We replace T_i with T_i' such that $P_i' = p_i \cdot P_i$ and $C_i'^j = C_i^j$ for $0 \leq j \leq N_i - 1$, and we increase C_n^0 by the amount needed to again extremely utilize the processor. This increase is smaller than $C_i^1 \cdot (p_i - 1)$. Let the old and new utilization factors be U^m and U'^m respectively.

$$\begin{aligned} U'^m &\leq U^m + (p_i - 1) \cdot C_i^0/P_n + C_i^0/P_i' - C_i^0/P_i \\ &= U^m + C_i^0 \cdot (p_i - 1)/(1/(p_i \cdot P_i + q_i) - 1/(p_i \cdot P_i)) \\ &\leq U^m \end{aligned}$$

Therefore we can conclude that the minimum utilization occurs among task sets in which the longest period is no larger than twice of the shortest period. This establishes Lemma 5. **QED**

Theorem 4 *Let $r = min_{i=1}^{n}(C_i^0/C_i^1)$. For task sets of size n, the utilization bound is given by $r \cdot n \cdot ((\frac{r+1}{r})^{1/n} - 1)$.*

Proof. By definition, the least upper bound is the minimum of the $\overline{U^e}$ for task sets of size ranging from 1 to n, and we have $min_{i=1}^{n}(r \cdot i \cdot ((\frac{r+1}{r})^{1/i} - 1)) = r \cdot n \cdot ((\frac{r+1}{r})^{1/n} - 1)$. **QED**

We observe that Liu and Layland's Theorem 1 is a special case of Theorem 4 with $r = 1$ and the frame separation time equals the period.

The following tables summarize the relative advantage of using the multiframe model over the L&L model in determining whether a set of task is scheduleable. The column under $U_{L\&L}$ gives the utilization bound in the L&L model.

	$U_{L\&L}$	r=2	3	4	5	6	7	8	9	10	∞
n=2	0.828	8.5	12.0	14.0	15.2	16.1	16.7	17.2	17.5	17.8	20.7
3	0.780	11.4	16.2	18.8	20.5	21.7	22.6	23.2	23.8	24.2	28.2
4	0.757	12.8	18.2	21.3	23.2	24.6	25.6	26.4	27.0	27.4	32.1
5	0.743	13.6	19.5	22.8	24.9	26.3	27.4	28.2	28.9	29.4	34.5
10	0.718	15.3	22.0	25.8	28.2	29.9	31.1	32.1	32.8	33.4	39.3
20	0.705	16.2	23.3	27.3	29.8	31.6	33.0	34.0	34.8	35.5	41.8
30	0.701	16.4	23.7	27.8	30.4	32.2	33.6	34.6	35.5	36.1	42.6
40	0.699	16.6	23.9	28.0	30.7	32.5	33.9	35.0	35.8	36.5	43.0
50	0.698	16.7	24.0	28.2	30.8	32.7	34.1	35.2	36.0	36.7	43.3
100	0.696	16.8	24.3	28.5	31.2	33.1	34.5	35.5	36.4	37.1	43.8
∞	0.693	17.0	24.5	28.8	31.5	33.4	34.9	35.9	36.8	37.5	44.3

Table 1: Utilization Bound Percentage Improvement

Table 1 shows the percentage improvement of our bound over the Liu and Layland bound. Specifically, the table entries denote $100 * (\overline{U^m}/U_{L\&L} - 1)$, for different combination of r (the ratio of the peak execution time to the execution time of the second frame) and n (the number of tasks in the task set). For example, suppose we have a system capable of processing one Gigabyte of data per second, and a set of tasks each of which needs to process one Megabyte of data per second. Using a utilization bound of $\ln 2$, we can only allow 693 tasks. By Theorem 4, we can allow at least 863 tasks (over 24% improvement) when $r \geq 3$.

As r increases, the bound improvement increases. Actually, as $r \to \infty$, a simple calculation shows that the bound $\to 1$. This says that our model excels when the execution time of the task varies sharply.

It is also interesting to compare the maximum average utilization with L&L bound. However, the maximum average utilization factor may be arbitrarily low even if the maximum utilization factor is very high. One simple example is $\{((10, 5, 1, 1, \ldots, 1), 10)\}$. So, we take instead the average of the first two frames of the task. In Table 2 we calculate $100 * (\frac{1}{2}(U^m + \frac{U^m}{r}))/U_{L\&L}$, which is the ratio of the biggest possible maximum average utilization factor to Liu and Layland bound. Table 3 shows we can still maintain good overall system utilization when task execution time varies.

4 Conclusion and Future Research

In this paper, we give a multiframe model for real-time tasks which is more amenable to specifying tasks whose execution time varies from one instance to another. In our model, the execution times of successive instances of a task is specified by a finite array of numbers rather than a single number which is the worst-case execution time of the classical Liu and Layland model.

Using the new model, we derive the utilization bound for the preemptive fixed priority scheduler, under the assumption that the execution time array of the tasks satisfies the AM (Accumulative

	$U_{L\&L}$	r=2	3	4	5	6	7	8	9	10	∞
n=2	0.828	81.4	74.7	71.2	69.1	67.7	66.7	65.9	65.3	64.8	60.4
3	0.780	83.5	77.4	74.3	72.3	71.0	70.0	69.3	68.8	68.3	64.1
4	0.757	84.6	78.8	75.8	73.9	72.7	71.8	71.1	70.5	70.1	66.1
5	0.743	85.2	79.7	76.7	74.9	73.7	72.8	72.1	71.6	71.2	67.3
10	0.718	86.5	81.3	78.6	76.9	75.8	74.9	74.3	73.8	73.4	69.7
20	0.705	87.1	82.2	79.5	77.9	76.8	76.0	75.4	74.9	74.5	70.9
30	0.701	87.3	82.4	79.9	78.2	77.1	76.3	75.7	75.3	74.9	71.3
40	0.699	87.4	82.6	80.0	78.4	77.3	76.5	75.9	75.4	75.1	71.5
50	0.698	87.5	82.7	80.1	78.5	77.4	76.6	76.0	75.6	75.2	71.6
100	0.696	87.6	82.8	80.3	78.7	77.6	76.8	76.2	75.8	75.4	71.9
∞	0.693	87.7	83.0	80.5	78.9	77.8	77.1	76.5	76.0	75.6	72.1

Table 2: Ratio of maximum average to L&L bound

Monotonic) property. This property is rather liberal and is consistent with common encoding schemes in multimedia applications, where one of the execution times in an array "dominates" the others. We show that significant improvement in the utilization bound over the Liu and Layland model results from using our model. This is useful in dynamic applications where the number of tasks can vary and the figure of merit for resource allocation is the number of tasks that the system can admit without causing timing failures.

Work is under way to apply this model to real-life applications such as video stream scheduling and will be reported in the future.

References

[1] C. L. Liu and James W. Layland, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, Journal of ACM, Vol. 20, No. 1, January 1973.

[2] Al. Mok, *Fundamental Design Problems of Distributed systems for the Hard-Real-Time Environment*, Ph.D. Thesis, MIT, 1983

[3] Tei-Wei Kuo and Aloysius K. Mok, *Load Adjustment in Adaptive Real-Time Systems*, IEEE 12th Real-Time Systems Symposium, December 1991.

[4] J. Lehoczky, L. Sha, and Y. Ding, *The Rate Monotonic Scheduling Algorithm - Exact Characterization and Average Case Behavior*, Proceedings of the IEEE Real-Time System Symposium, 1989

[5] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, *Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems*, Technical Report CS-94-01, University of Virginia, Computer Science Department, January 1994

[6] Jay K. Strosnider, John P. Lehoczky, and Lui Sha, *The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments*, IEEE Transactions on Computers, Vol. 44, No. 1 January 1995

[7] A. Burns and A. J. Welling, *Dual Priority Assignment: A Practical Method for Increasing Processor Utilization*, Proceedings of Fifth Euromicro Workshop on Real-Time Systems, Oulu, pp. 48-55, 1993

[8] T. M. Ghazalie, T. P. Baker, *Aperiodic Servers in a Deadline Scheduling Environment*, Real-Time Systems, Vol. 9, No. 1, July 1995

[9] B. Sprunt, L. Sha, and J. Lehoczky, *Aperiodic Task Scheduling for Hard Real-Time Systems*, Real-Time Systems: The International Journal of Time-Critical Computing Systems, Vol. 1, pp. 27-60, 1989

[10] Jen-Yao Chung, J.W. S. Liu, and Kwei-Jay Lin, *Scheduling Periodic Jobs That Allow Imprecise Results*, IEEE Transactions on Computer, Vol. 39, No. 9, September 1990