A General 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 generalized real-time task model which allows the execution time of a task to vary from one instance to another by capturing the maximum total execution time of consecutive instances. We investigate the scheduleability problem for this model with the help of the multiframe task model we introduced in [4]. We show that a significant improvement in the utilization bound can be established for the general model and the requirement of AM property in the multiframe model can be dropped.

[*]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 [5] 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 [7, 8, 9, 10]. In [11], 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.

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

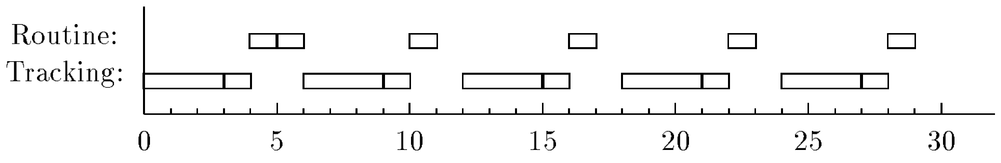

Figure 1: Schedule of the Vehicle Tracking System

the task is to be executed every 5 time units. Obviously, the computer cannot handle both tasks. 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.

To address real-time task with varying execution times in general, we first proposed in [4] a *multiframe task model* in which task execution time follows a known pattern. We showed that better scheduleability bounds can be obtained. We are applying this model to multimedia analysis.

In this paper, we further drop the pattern requirement and investigate the scheduleability of task whose execution time varies arbitrarily. Our generalized task model only makes use of the maximum total execution time of consecutive requests, no matter when and where it happens. We shall establish the utilization bounds for our model which will be shown to subsume the L&L result [1]. It will be seen that the scheduleability bounds can be improved substantially if there is a large variance in the execution time of a task. Using the generalized model, we can safely admit more real-time tasks than the L&L model. We shall also note that the AM constraint for multiframe task model disappears in the general model.

The paper is organized as follows. Section 2 introduces our general real-time task model. Section 3 recaps the multiframe task model. Section 4 investigates the relation of the general task model to the multiframe model and new results for both models are derived. Section 5 is the conclusion.

2 The General 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 general real-time task is a tuple* (Φ, P)*, where* Φ *is an array of execution times* $(\phi^1, \phi^2, \ldots)$*, 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 sum of any* i $(i > 0)$ *consecutive execution times of the task is no larger than* ϕ^i*, the ith element of* Φ*. The deadline of each frame is* P *after its ready time.*

Example 1 *The vehicle tracking system mentioned at the beginning of this paper can be modelled as* $\{(\Phi_1, 3), (\Phi_2, 5)\}$ *where* $\Phi_1 = (3, 4, 7, 8, \ldots)$, $\Phi_2 = (1, 2, 3, 4, \ldots)$.

For task with constant execution time C, $\Phi = (C, 2C, 3C, 4C, \ldots)$. Apparently, the more elements in Φ we can make use of, the better the chance of task feasibility. The L&L model only make use of the first element in Φ and assumes all frames have execution time ϕ^1.

Later in the paper, we will call the general model task just task when there is no confusion. Other specific model task will be called with its model name, such as L&L task or multiframe task.

Corollary 1 *The average execution time of a task* (Φ, P) *is* $lim_{i \to \infty} \frac{\phi^i}{i}$

To analyze the generalized task, we presume the knowledge of all elements of Φ. Unfortunately, it is not always possible practically. The following lemma solves this problem.

Lemma 1 *For task* (Φ, P) *and any integer* m *and* n, *Let* $n = p * m + q$, $0 \leq q < m$. *The maximum sum of* n *consecutive execution times is no larger than* $p * \phi^m + \phi^q$. *Let* $\phi^q = 0$ *if* $q = 0$.

Proof. For any n consecutive frames of the task, we divide them into $p + 1$ groups with every m frames a group and the last group contains q frames. By definition, All those p-frames groups have total execution time no more than ϕ^m, and the last group has execution no more than ϕ^q. So the total of these n frame execution times is no more than $p * \phi^m + \phi^q$. **QED.**

For a task with first m known elements of Φ, the best estimate for ϕ^k(here $k > m$) is $min_{1 \leq i \leq m}(p_i * \phi^i + \phi^{q_i})$, where $k = p_i * i + q_i$, $0 < q_i < i$. Note we are using the minimum instead of $p_m * \phi^m + \phi^{q_m}$ simply because the minimum may not happen in the latter. For example, with $\Phi = (4, 6, 9, \ldots)$, the best estimate for ϕ^{10} is 30 when $i = 2$, not 31 when $i = m = 3$.

With lemma 1 in mind, we will later assume that every element in Φ is available.

In the proofs to follow, we shall often associate a task with a periodic task in the L&L model which has the same execution time as the maximum frame time and whose period is the same as the minimum separation of the multiframe task. Consider a task $T = (\Phi, P)$, its corresponding L&L task is (ϕ^1, P). We shall call ϕ^1 the *peak execution time* of task T.

Definition 2

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

We call $U^m = \Sigma_{i=1}^{n} \phi_i^1 / P_i$, the peak 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.

A pessimistic way to analyze the scheduleability of a 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* barely *utilizing the processor if it is scheduleable by A, but increasing the execution time of some frame of some 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 barely utilizing task sets with respect to the scheduling policy A.

Lemma 2 *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 L&L task set whose utilization factor is $\overline{U_A^m} > 1$. Since a L&L task is just a special case of a general task, this set is supposed to be scheduleable by the definition of $\overline{U_A^m}$. But obvoiusly this is not possible. So our assumption is wrong. So $\overline{U_A^m}$ cannot exceed 1. **QED**.

Lemma 3 *Suppose A is a scheduling policy which can be used to schedule both general and L&L task sets. Let the utilization bound of A be $\overline{U_A^m}$ for generalized 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 first deadline missing happens with ith frame of task T_j 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 3 strict? Intuitively, if U^m of a task set is larger than $\overline{U_A^c}$ and there is not much variance in the execution times of the tasks in the set, the task set is unlikely to be scheduleable. However, if the variance is sufficiently big, the same scheduling policy will admit more tasks. This can be quantified by determining the utilization bound for our task model.

Definition 4 *The worst running case of a task* (Φ, P) *is such that it requests at the maximum rate and the total execution time of its first i frames is* ϕ^i *for any* $i > 0$*. In other words, all consecutive frames are separated by* P *time units, and the first frame has execution time* ϕ^1*; the ith frame has execution time* $\phi^i - \phi^{i-1}$ *for* $i > 1$*.*

Definition 5 *The critical instance of a task is the period when its peak execution time is requested simultaneously with all higher priority tasks, and all higher priority tasks have worst running cases.*

It turns out that the critical instance is the worst case for a task. In other words, it has the longest finish time in the critical instance.

Theorem 1 *A task (Φ, P) is scheduleable by a fixed priority scheduler if it passes the critical instance test.*

Proof. Suppose a task $T_k = (\Phi_k, P_k)$ 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^1_{end}. In other words, $t_{end} \leq t^1_{end}$. 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^1_{end}, say t^2_{end}. In other words, $t^1_{end} \leq t^2_{end}$. Now, we re-assign the execution time of all higher priority tasks such that they run the worst case starting at time t_0. This re-assignment has the effect of postponing(if any) the finish time of T_k to t^3_{end}, $t^2_{end} \leq t^3_{end}$. At last, we change T_k's execution time to be ϕ^1_k. This further(if any) postpones the finish time to t^4_{end}, $t^3_{end} \leq t^4_{end}$. By construction, the resulting request pattern is the critical instance for T_k. Since T_k is scheduleable in its critical instance, we have $t^4_{end} - t_0 \leq P_k$, 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^1_{end} where $t_{end} \leq t^1_{end}$. 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^1_{end}, say t^2_{end}. In other words, $t^1_{end} \leq t^2_{end}$. Now, we re-assign the execution time of all higher priority tasks such that they run the worst case starting at time t_0. This re-assignment has the effect of postponing(if any) the finish time of T_k to t^3_{end}, $t^2_{end} \leq t^3_{end}$. At last, we change T_k's execution time to be ϕ^1_k. This further(if any) postpones the finish time to t^4_{end}, $t^3_{end} \leq t^4_{end}$. This last case is actually the critical instance for T_k. Since T_k is scheduleable in its critical instance, we have $t^4_{end} - t_0 \leq P_k$, 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**.

Example 2 *The task set mentioned in example 1 is scheduleable because it passes the critical instance test. However, the corresponding L&L task set* $\{(3,3),(1,5)\}$ *with utilization factor* 1.2 *is obviously unscheduleable by any scheduling policy.*

Theorem 2 *If a feasible priority assignment exists for some 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**.

We are now ready to analyze the utilization bound for fixed priority scheduling. But before that, we will stop here and introduce in the next section a multiframe task model. With the result derived from multiframe task model, we shall come back in section 4 and prove a higher utilization bound for the generalized task model.

3 The Multiframe Task Model

We will use the result of the multiframe task model [4] to derive a higher utilzation bound for the generalized task model. To make this paper self-consistent, we will provide in this section necessary infomation about the multiframe model. Also to save space, we leave out some not too difficult parts. Please note that the multiframe task model itself is a complete model and has special properties that are used by us to analyze multimedia scheduling. Please see [4] for more detail.

Definition 6 *A multiframe real-time task is a tuple* (Γ, P)*, where* Γ *is an array of* N *execution times* $(C^0, C^1, \ldots, C^{N-1})$ *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* $C^{((i-1) \bmod N)}$*, where* $1 \leq i$*. The deadline of each frame is* P *after its ready time.*

Example 3 *the vehicle tracking system mentioned before can also be modelled as a multiframe task set* $S = \{T_1, T_2\} = \{((3,1),3),((1),5)\}$*. Its corresponding L&L task set* S' *is still* $\{(3,3),(1,5)\}$*. The peak utilization factor of* S *is still* $U^m = 1.2$*. The maximum average utilization factor of* S *is* 0.867*.*

To derive the utilization bound, we require that all multiframe tasks satisfy following AM property.

Definition 7 *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.*

Name	Pattern	Frames	Max-I(bits)	Max-P	Max-B	Average	Max-I/Max-B
bike.mpeg	IBBPBB	150	116288	75752	26184	34270	4.441

Table 1: Statistics of a Video Script from Terminator-I

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 example, all tasks in Example 3 are AM. As another example, tasks in multimedia applications usually satisfy this restriction. The well known MPEG(Moving Picture Experts Group) [12] has defined a standard to encode video. There are three kinds of video frames in MPEG: I-frame, P-frame, and B-frame. I-frame uses intra-frame encoding. P-frame and B-frame use inter-frame encoding. P-frame is encoded with the data of its own and its previous I-frame or P-frame. B-frame is encoded with the data of its own and both its previous and following I(P)-frames. Between any two successive I-frames there are a fixed number of P-frames; Between any two successive I- or P-frames there are also a fixed number of B-frames. In other words, the video frame sequence follows a pattern such as "IBBPBBPBBIBB...". Generally the size of an I-frame is larger than that of a P-frame, and the size of an I-frame or a P-frame is much larger than that of a B-frame. Table 1 lists the statistics of a video clip from the movie "Terminator-I" when Arnold Schwarznegger is riding a motor cycle. This video clip can be modelled as a multiframe AM task $\{(116288, 34270, 34270, 75752, 34270, 34270), T\}$ where T is the inverse of frame frequency, as is typically the case.

The utilization bound for the L&L model is given by the following theorem in the much cited paper [1].

Theorem 3 (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 8 *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 4 ([4]) *For the preemptive fixed priority scheduling policy, a multiframe task is scheduleable if it is scheduleable in its critical instance.*

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

Corollary 2 *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. Thinking about the critical instance test, it is clear that such a task replacement does not affect the result of the critical instance test.

Note that a multiframe task (Γ, P) is a special case of a general task (Φ, P) with $\phi^i = p * \Sigma_{j=0}^{N-1} C^j + \Sigma_{j=0}^{q-1} C^j$, where $i = p * N + q,\ 0 \leq q < N$.

Example 4 *Again, the vehicle tracking system modelled in example 3 is scheduleable by RMA since it is AM and passes the critical instance test*

Lemma 4 ([4]) *If a feasible priority assignment exists for some multiframe task set, the rate-monotonic priority assignment is feasible for that task set.*

To compute the utilization bound for multiframe tasks, we need the following lemma. The idea of this lemma comes from [6].

Definition 9 *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 5 $\Psi(n, \alpha) = n \cdot (\alpha^{1/n} - 1)$.

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.**

We will use the result of Lemma 5 in Lemma 6.

Definition 10 *A multiframe task set is said to be extremely utilizing the processor if it is scheduleable but increasing the* 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 the utilization factors of all extremely utilizing multiframe task sets.

In the following, we shall adopt the convention $C_i^0/C_i^1 = 1$ if a multiframe task has only one execution time.

Lemma 6 *Consider all multiframe 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 4 and lemma 4, we only need to consider the case where all multiframe 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 of a task may be involved in all the critical instance tests.

First, we note that the utilization bound corresponds to the case where the ratio C^0/C^1 of every multiframe task equals r, since we can increase C^1 without changing U^m, and increasing C^1 will only take more CPU time. So without loss of generality we assume that all the C^0/C^1 ratios are equal to r.

For any scheduleable and extremely utilizing multiframe 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 multiframe 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 multiframe 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 $C_i^0 \neq 0$, 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 multiframe 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 multiframe 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 multiframe 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 multiframe 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 multiframe 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 multiframe 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 P_{i+1}.

From these four claims and Lemma 5, we can conclude:

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

Finally, we note that in the case where $P_i = (\frac{r}{r+1})P_n$, we can safely transfer all the execution time of task T_i to T_n without invalidating the previous argument. **QED**

Lemma 7 *Let $r = min_{i=1}^{n}(C_i^0/C_i^1)$. For multiframe 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 multiframe tasks request at the maximum rate. For any task T_i in an extremely utilizing multiframe 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 $C_i^0 \cdot (p_i - 1)$. 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 multiframe task sets in which the longest period is no larger than twice of the shortest period. This establishes Lemma 7.

QED

Theorem 5 *Let $r = min_{i=1}^{n}(C_i^0/C_i^1)$. For multiframe 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**

4 Fixed Priority Scheduling

In the last section we proved that there is a much better utilization bound for the multiframe task model. We will show in this section that the same result applies to the general task model.

Definition 11 *For a general task $T_g = (\Phi, P)$, define its corresponding multiframe task w.r.t. an integer n to be $T_m = (\Gamma, P)$, where $\Gamma = (\phi^1, \phi^2 - \phi^1, \ldots, \phi^n - \phi^{n-1})$.*

Lemma 8 *The total execution time of i ($i > 0$) consecutive frames of a general task is no more than the total execution time of the first i frames of its corresponding multiframe task.*

Proof. It is easy to calculate that for the task $T_g = (\Phi, P)$, the total execution time of the first i frames of its corresponding multiframe task is ϕ^i if $i \leq n$ or $p * \phi^n + \phi^q$ if $i > n$, where $i = p * n + q$ $(0 \leq q < n)$. By definition and lemma 1, this is the upper bound. **QED**.

Lemma 9 *For a task in a task set, if its corresponding multiframe task passes its criticail instance test in the corresponding multiframe task set, then the original task also passes its critical instance test in the original task set*

Proof. Suppose a task's corresponding multiframe task is scheduleable but itself is unscheduleable. Let's say it misses deadline at time t in the critical instance test. We now substitute each task T_g with its corresponding multiframe task T_m. According to lemma 8, T_m will have no less pending execution time than T_g at any time in $(0, t)$. This makes the situation worse. So after substitute all random tasks with their corresponding multiframe tasks, we still have a unscheduleable situation. This contradicts that the corresponding multiframe task is scheduleable. So if its corresponding multiframe task is scheduleable by the scheduling policy, the original task is scheduleable in the original task set **QED**.

Example 5 *A AM multiframe task* $T_m = ((4, 2, 3, 3, 3, 2), P)$ *can be modeled as a random task* $T_g = ((4, 6, 9, 12, 15, \ldots), P)$. T_g*'s corresponding multiframe task w.r.t.* $n = 3$ *is* $((4, 2, 3), P)$, *which is, unfortunately, not AM.*

Note in lemma 9 if all the multiframe task pass the critical instance test, we know the original task set will be feasible. However, the multiframe task set may still not be feasible because some multiframe task may not be AM.

Theorem 6 *A set of tasks is scheduleable by a scheduling policy if its corresponding multiframe task set is scheduleable by the same scheduling policy.*

Proof. Suppose its corresponding multiframe task set is scheduleable but itself is unscheduleable. Let's say some task miss deadline at time t. We go backward from t until hit a point t_0 when CPU is idle. t_0 always exists since at time 0 no task is scheduled. We now substitute each task T_g with its corresponding multiframe task T_m such that the first frame of T_m in the time frame (t_0, t) is the maximum frame. According to lemma 8, T_m will have no less pending execution time than T_g at any time in (t_0, t). This makes the situation worse. So after substitute all random tasks with their corresponding multiframe tasks, we still have a unscheduleable situation. This contradicts that the corresponding multiframe task set is scheduleable. So if its corresponding multiframe task set is scheduleable by the scheduling policy, the random task set is scheduleable **QED**.

Note theorem 5 does not require if the scheduling policy has fixed or dynamic priority assignment.

Theorem 7 *For random task sets of size n, let $r = min_{i=1}^{n}(\phi_i^1/(\phi_i^2 - \phi_i^1))$. The utilization bound is given by $r \cdot n \cdot ((\frac{r+1}{r})^{1/n} - 1)$.*

Proof. For a task $T_g = ((\phi^1, \phi^2, \ldots), P)$ in the task set, let's transform it into a multiframe task w.r.t. integer 2: $T_m = ((\phi^1, \phi^2 - \phi^1), P)$. It is trivial to see that T_m is AM. According to theorem 5, if $\Sigma_{i=1}^{n}\phi_i^1 \leq r \cdot n \cdot ((\frac{r+1}{r})^{1/n} - 1)$, the corresponding multiframe task set is scheduleable by a fixed priority scheduler. Therefore, according to theorem 6, the random task set is scheduleable. **QED**.

	$U_{L\&L}$	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 2: Utilization Bound Percentage Improvement

We observe that Liu and Layland's Theorem 3 is a special case of Theorem 7 with $r = 1$ and the frame separation time equals the period.

Table 2 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 and n mentioned in theorem 7. 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

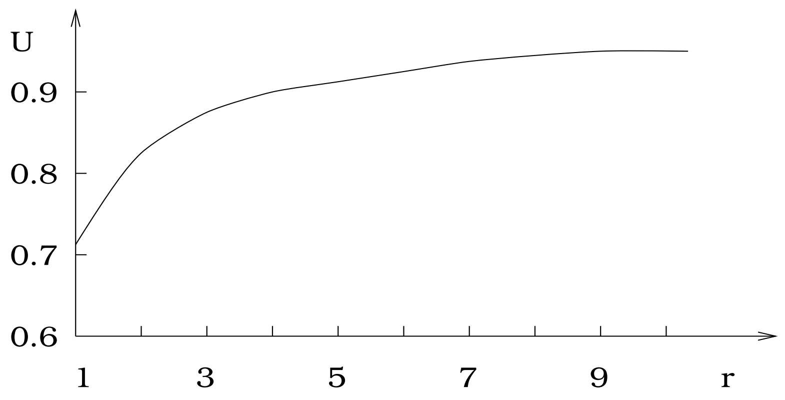

Figure 2: Utilization Bound with n=10

data per second. Using a utilization bound of $\ln 2$, we can only allow 693 tasks. By Theorem 7, 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 $\to 1$. Figure 2 plots the bound against r when there are 10 tasks. This graph supports the observation that our model excels when the execution time of the task varies sharply from frame to frame.

Previously we observed that tasks in multimedia applications varies significantly frame by frame. We randomly select 20 MPEG files. The distribution of their r is shown in Figure 3. In all cases, r is bigger than 1.

Note in theorem 5, the utilization bound for fixed priority is derived on the condition that the multiframe task satisfies AM property. This can be removed if we treat the multiframe task as a special case of random task and determine its r by that in theorem 7.

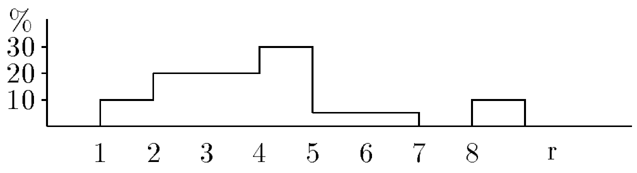

Figure 3: Distribution of max(I)/max(B) of 20 MPEG Videos

5 Conclusion

In this paper we proposed a generalized task model which allows a task's execution time to vary arbitrarily. Our model does not require any special constraint on the tasks. The more the task execution time varies, the more feasible a task set will be. A much higher utilization bound for fixed priority scheduling is achieved by our model.

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] Aloysius K. Mok and Deji Chen, *A Multiframe Model for Real-Time Tasks*, IEEE 17th Real-Time Systems Symposium, December 1996.

[5] 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

[6] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, *Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems*, IEEE Transactions on Computers, vol. 44, no. 12, pp 1429-1442, December 1995.

[7] 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

[8] 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

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

[10] 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

[11] 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

[12] D. Le Gall *MPEG: A Video Compression Standard for Multimedia Applications* Communications of the ACM, Vol. 34, No. 4, P46-58, April 1991