Real-Time Scheduling of Multimedia Tasks*

Aloysius Mok† Sanjoy Baruah⋆ Deji Chen† Sergey Gorinsky†

† Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
{*mok, cdj, gorinsky*}@cs.utexas.edu

⋆ Department of Computer Science
The University of Vermont
Burlington, VT 05405, USA
sanjoy@cs.uvm.edu

Abstract

Emerging multimedia applications require real-time delivery of information over computer networks. Providing such timeliness guarantees in a dynamic environment needs an efficient admission test for data transmission requests. Real-time scheduling theory has produced a number of efficient procedures for testing schedulability. Unfortunately, most of the solutions are derived in the context of task models where each instance of a task takes constant execution time. When applied to the multimedia domain, these models fail to characterize bursty multimedia data precisely. As a result, resources are used ineffectively. To address this problem and to exploit the benefits of existing solutions, this paper presents two task models – Enhanced Frame Model (EFM) and MultiFrame Model (MFM). The problem of transforming a periodic task system to an EFM task system is considered and its computational complexity is studied. An algorithm for the transformation is derived. The MFM model can be applied when the internal patterns of tasks are known and is shown to provide much better utilization bounds. Experiments with real-time video transmission are performed to evaluate usefulness of the proposed models.

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

1 Introduction

The necessity to support real-time delivery of multimedia data in the dynamic environment of computer networks makes it essential to process new data transmission requests quickly and effectively. This raises a demand for efficient admission tests.

Real-time scheduling theory thoroughly explored schedulability testing based on resource utilization bounds. In the famous paper on hard real-time periodic tasks [9], Liu and Layland derived bounds for the Earliest Deadline First (EDF) scheduling discipline and fixed-priority scheduling policies. Lehoczky [8] obtained a utilization bound for fixed-priority scheduling of task systems with arbitrary deadlines. Kuo and Mok [6] introduced the concept of a harmonic chain to achieve a better utilization bound.

Bound-based tests can be implemented in constant time per new transmission request. This makes them a very promising approach to be used in request admission control. However, most of the bound-based tests were obtained for task models with constant execution time per each instance of a task. In the real world, execution time of different task instances varies. Using the maximum value as the task execution time preserves performance guarantees but results in inefficient resource utilization. Such task modeling may have drastic consequences in the multimedia domain where bursty data streams are quite common (For example, in the MPEG (Moving Picture Experts Group) standard, video information is compressed into sequences of variable-size frames with the largest I-frame being ten times as big as the smallest B-frame [2, 7]). If the maximum utilization factor of a task system is twice its average utilization factor, only half of the resources can be used by the real-time task system, the other half may be just wasted.

A number of approaches have been investigated to cope with this problem. One of them is to use allocated but not utilized resources for scheduling non-real-time tasks [3]. Unfortunately, this solution cannot increase the number of admitted real-time tasks. The idea behind imprecise computation [1] is that execution time can be traded for precision of the task. Resources are allocated to guarantee some minimal performance requirements. When overflow occurs, shorter versions of tasks are scheduled. Although able to improve resource utilization, this scheme does not suit many applications for which reduction of execution times is not acceptable. Another way of dealing with variable execution times is to reduce the variation itself. Lam, Chow, and Yau [7] tried to smooth fluctuation of video streams but the resulting maximum transmission rate was still thrice as large as the minimum rate.

The discussed approaches attempted to solve the problem of inefficient resource utilization without addressing its source — inability of existing models to characterize tasks with variable execution time precisely. This paper refers to such tasks as irregular tasks and presents two models for their description. By combining these models with bound-based schedulability testing, we can obtain fast admission tests and efficient utilization of resources.

The paper is structured as follows. Section 2 formally introduces the notion of an irregular task. A task buffering technique is discussed and the Enhanced Frame Model is presented. Section 3 studies the

Multiframe Model which was introduced in [11]. This model is applicable when the internal structures of multimedia streams are known and provides much better utilization bounds. Experimental results are given in Section 4. Conclusions are provided in Section 5.

2 Irregular Tasks and The Enhanced Frame Model

Multimedia data are bursty by nature. When they are modeled as regular tasks with constant execution times, system resources are wasted because of imprecise data characterization. To address this problem, we introduce a notion of irregular tasks which takes into account the fact that different instances of a task may need different execution times. To preserve consistency with previous work [11], we refer to a task instance as a *frame*.

Definition 1 *An* **irregular periodic real-time task** *is a tuple* (Γ, P, D)*, where* Γ *is a set of execution times,* P *is the period of the task, and* D *is its deadline. The execution time of each frame (instance) of the task belongs to* Γ*.*

Unless specified otherwise, we will mean an irregular task when referring to a task. We will call a task with constant execution time a regular task.

Although fixed-priority scheduling was a starting point for this work, we do not restrict our analysis to any specific scheduling discipline. Unless stated explicitly, obtained results do not rely on any particular scheduling policy. The only condition imposed on the scheduling discipline is existence of a bound-based test. Testing based on resource utilization bounds can be implemented very efficiently which makes it particularly suitable for admission control.

The maximum utilization factor U^{max} of an irregular task may be much larger than its average utilization factor U^{ave}. When the maximum utilization factor of an irregular task system does not exceed the utilization bound, the requested performance can be guaranteed. If the average utilization factor is larger than the bound, the employed bound-based method cannot confirm schedulability of the system. The most interesting case happens when the bound of the scheduling policy is larger than the average utilization factor of the task system but smaller than its maximum utilization factor. Refusing to admit such systems ensures that no granted guarantee will be violated. On the other hand, this solution results in low resource utilization.

Our goal is to improve utilization of resources while preserving performance guarantees. Below we present task buffering which can be viewed as a characterization/performance negotiation technique. If the deadline of a task is postponed, the maximum utilization factor can drop below the testing bound. As a result, the bound-based test will recognize schedulability of the task system which was previously considered unschedulable. However, this method has its limitations. First, deadlines cannot be extended further than some maximum values; otherwise, the task system is not real-time and becomes schedulable

by selecting infinite deadlines (if the average utilization is smaller than the testing bound). Second, when deadlines are postponed, additional buffers are needed to store pending frames. Thus, one has to make sure that sufficient space for task buffering is always available.

When these restrictions are taken into account, our problem becomes as follows: *given an unschedulable irregular task system, which deadlines should be assigned to every task so that the deadlines do not exceed the requested values, there is enough buffer space, and the system becomes schedulable?*

Before presenting our solution, we want to comment on choosing new values for task deadlines. It was shown that analysis for arbitrary postponed deadlines is much more complicated than for multiple-period deadlines [8]. Since restricting deadlines to multiples of task periods simplifies the analysis and retains the ability to improve resource utilization, we require each postponed deadline be equal to a multiple of the corresponding task period.

When the deadline is postponed, irregularity of the task can be smoothed since consecutive frames of the task are combined to form a larger frame which we refer to as an *enhanced frame*. Because of this reduction of the maximum utilization factor, the task system can become schedulable. We transform each irregular task T with period P and deadline $D = P$ into an enhanced-frame task T' with period $k \cdot P$ where k is the number of consecutive frames we want to combine for the given task. The execution time of T' is the sum of k consecutive execution times of T; the maximum execution time of T' may be much smaller than k times the maximum execution time of T. The deadline for each enhanced frame is $k \cdot P$. We will call this task model the Enhanced Frame Model. A more formal and general definition of the *Enhanced Frame Model (EFM)* is as follows:

Definition 2 *Let* (Γ, P, D) *be an irregular periodic task. An* **enhanced-frame task** *is a tuple* (Γ', P', D') *where* Γ' *is a set of execution times of enhanced frames,* $P' = \lambda(P)$ *is a period of enhanced frames,* $D' = \mu(D)$ *is a deadline of each enhanced frame. We refer to* $\lambda(\cdot)$ *as a* **frame-enhancement function***;* $\mu(\cdot)$ *is called a* **deadline-postponement function**.

In the considered transformation of an irregular task, we use identical frame-enhancement and deadline-postponement functions $\lambda(x) = \mu(x) = k \cdot x$ and $\Gamma' = \{A^k\}$ where k is a positive integer and A^k is the maximum cumulative execution requirement of k consecutive original frames. To be consistent with previous work [4], we refer to A^k as an *arrival bound function* of the task. The arrival bound function affects the quantitative gain from transforming frames to enhanced frames and can be determined from constraints imposed on the task. For example, if execution requirement of the task is characterized by $(C^{max}, C^{ave}/P)$ leaky bucket model (i.e. any k successive frames may not require more than C^{max} additional execution time over the average rate C^{ave} per frame), then the arrival bound function equals to $A^k = C^{max} + k \cdot C^{ave}$.

Note that depending on the position inside the enhanced frame, an original frame is guaranteed at least one of the following delay bounds: $k \cdot P$, $k \cdot P + 1$, ..., $(2k - 1)P$. For instance, if the enhanced

frame consists of three original frames, its first frame is guaranteed a delay bound of $5P$, the second one – $4P$, and the third frame – $3P$.

Let us consider an architecture where the number of buffers is constant and each frame is stored in a separate buffer.

First, we compare the EFM with the Arbitrary Deadline Model (ADM) when fixed-priority scheduling is used. The ADM is introduced by Lehoczky [8] and generalizes the regular task model by allowing tasks to have arbitrary deadlines. If deadlines D is a multiple of periods P and $\Delta = D/P$, the utilization bound approximates $U_{AD} = \Delta \ln \frac{\Delta+1}{\Delta}$ when the number of tasks grows to infinity. The number of required buffers equals to $\Delta + 1$ per task (the extra buffer is for storing the frame being served). Since U_{AD} exceeds $\ln 2$ (the bound for regular task systems with $D = P$), employing the ADM can improve resource utilization. However, this model assumes constant execution time; in the case of bursty tasks such as multimedia streams, the utilization still remains very low. Besides, when Δ grows, the ADM bound improves slower (see Figure 1 in [8]) and is always less than 1. Whenever the maximum utilization factor of a task system exceeds 1, applying the ADM cannot make the system schedulable no matter how many additional buffers are provided. The EFM does not have this limitation. When the EFM is employed, k frames are combined into an enhanced frame, and the new maximum utilization factor becomes $U_{EF} = \Sigma(A_i^k/(k \cdot P_i))$ where A_i^k denotes the arrival bound function of task i. Increasing k always results in reducing the maximum utilization factor. As a matter of fact, when $k \to \infty$, the task system becomes schedulable since U_{EF} approaches the average utilization which is smaller than $\ln 2$ (the testing bound for fixed-priority scheduling of regular tasks). Since only a constant number of frame buffers is available, k is not allowed to become arbitrary large. When the EFM is employed, $2k$ buffers are necessary per task: k buffers for the enhanced frame being served, and the other k buffers for the arriving enhanced frame (without deadline postponement, each task needs two buffers: one for the frame being processed, and another for the arriving frame).

Taking into account that the amount of buffers is constant and that delays are not allowed to exceed some requested values, we formalize the studied problem as follows:

Problem 1 *Given an irregular task system* $S = \{T_i = (\Gamma_i, P_i, D_i) | 1 \le i \le n\}$, $D_i = P_i$, *arrival bound functions* A_i^k, *maximum allowed delays* d_i *where* $1 \le i \le n$, $2b$ *frame buffers, and utilization bound* U, *find a list* $(k_1, k_2, \ldots, k_n)$ *such that* $2k_i \cdot P_i \le d_i$ *for* $1 \le i \le n$, $\Sigma_{i=1}^n k_i \le b$, *and* $\Sigma_{i=1}^n (A_i^{k_i}/(k_i \cdot P_i)) \le U$.

Using a separate buffer for each frame can lead to inefficient memory utilization when frame sizes vary a lot. Many architectures, such as ATM switches, employ a fixed-size buffer space shared by all tasks. Below we formalize the studied problem for these configurations:

Problem 2 *Given an irregular task system* $S = \{T_i = (\Gamma_i, P_i, D_i) | 1 \le i \le n\}$, $D_i = P_i$, *arrival bound functions* A_i^k, *maximum allowed delays* d_i *where* $1 \le i \le n$, *buffer space of size* b, *and utilization bound* U, *find a list* $(k_1, k_2, \ldots, k_n)$ *such that* $2k_i \cdot P_i \le d_i$ *for* $1 \le i \le n$, $\Sigma_{i=1}^n A_i^{k_i} \le b$, *and* $\Sigma_{i=1}^n (A_i^{k_i}/(k_i \cdot P_i)) \le U$.

We prove that both the problems are as hard as the NP-complete Knapsack problem (See Appendix A for the definition of the Knapsack problem):

Theorem 1 *Problem 1 is NP-complete.*

Theorem 2 *Problem 2 is NP-complete.*

Proofs of the theorems are given in Appendix B and Appendix C.

Let us consider a set of video streams passing through an ATM switch. According to Theorem 2, no polynomial algorithm (unless $NP = P$) exists to detect what size should be chosen for the enhanced frame of each stream so that the requested service could be guaranteed.

Below we provide an approximation polynomial-time algorithm which solves Problem 1 and tests schedulability of an irregular task system. As it will be shown, this algorithm is effective in practice.

The algorithm employs a list of tuples (T, F, R) where R is utilization reduction per additional frame caused by enlarging the enhanced frame of task T by F additional frames. The list is sorted in non-increasing order of R; tuples with the same value of R are sorted in non-increasing order of F. Given a task T, procedure Enlist takes into account buffer and delay limitations, generates tuples for task T, and inserts these tuples into the list. Initially, enhanced frames of all tasks are selected to contain one frame. After creating the list by running Enlist on all tasks of the system, the algorithm searches for the first tuple (T_1, F_1, R_1) such that F_1 does not exceed the number of available buffers. If there is no such a tuple, the algorithm claims that the task system is unschedulable. Otherwise, the enhanced frame of task T_1 is enlarged by F_1 additional frames. This decreases the maximum utilization factor by $F_1 \cdot R_1$. Then all tuples corresponding to T_1 are updated by removing old entries from the list and running Enlist on T_1 again. This sequence of looking for the first suitable tuple and updating the maximum utilization factor, the enhanced-frame sizes, and the tuple list is repeated until the maximum utilization drops below the testing bound, or the list is empty, or no buffers remain available. In the former case, the task system is claimed to be schedulable and the algorithm returns the sizes of the enhanced frames for each task. In other cases the algorithm suggests that the system is not schedulable.

A more formal description of the algorithm is as follows:

Algorithm 1 Schedulability Testing and Construction of Enhanced Fames

input $S = \{T_i = (\Gamma_i, P_i, D_i) | 1 \le i \le n\}$: *irregular task system;*
A_i^k: *arrival bound functions,* $1 \le i \le n$;
d_i: *maximum allowed delays,* $1 \le i \le n$;
b: *half the number of frame buffers;*
U: *resource utilization bound;*

```
type  Tuples = record
                 Task: integer;
                 AdditionalFrames: integer; {number of additional frames}
                 RelativeDecrease: real; {utilization reduction per additional frame}
               end;

var   EnhancedFrameSize: array[1...n] of integer; {numbers of frames in the enhanced frames}
      Tuple: Tuples;
      List: list of Tuples;
            {sorted in non-increasing order of RelativeDecrease; for equal values of
             RelativeDecrease, it is sorted in non-increasing order of AdditionalFrames}
      UsedBuffers: integer; {half the number of used buffers}
      InitialUtilization: real; {initial maximum utilization factor of the task system}
      UtilizationDecrease: real; {total reduction of the maximum utilization factor}
      Task: integer;
      ExtraFrames: integer;

procedure Enlist(Task: integer; List: list of Tuples);
{generate tuples for Task and insert them into List}
  var   CurrentSize, TestedSize: integer;
        CurrentUtilization, NewUtilization: real;
        NewTuple: Tuples;
  begin
    CurrentSize := EnhancedFrameSize[Task]; {size of the current enhanced frame}
    CurrentUtilization:= A_Task^CurrentSize/(CurrentSize · P_Task);
       {maximum utilization factor for the current enhanced frame}
    TestedSize := CurrentSize + 1;
    while (TestedSize ≤ CurrentSize + b − UsedBuffers) and (2 · TestedSize · P_Task ≤ d_Task) do
          begin
            NewUtilization := A_Task^TestedSize/(TestedSize · P_Task);
              {maximum utilization factor for the tested enhanced frame}
            if NewUtilization < CurrentUtilization then
               begin
                 NewTuple.Task := Task;
                 NewTuple.AdditionalFrames := TestedSize − CurrentSize;
                 NewTuple.RelativeDecrease := (CurrentUtilization − NewUtilization)/NewTuple.AdditionalFrames;
```

```
                    insert NewTuple into List;
                  end;
              TestedSize := TestedSize + 1;
          end;
  end;

begin
  for Task := 1 to n do EnhancedFrameSize[Task] := 1;
    {initially, every enhanced frame consists of one frame}
  InitialUtilization := Σ_{i=1}^{n} (A_i^1 / P_i);
  UsedBuffers := n; {initially, 2n buffers are used}
  UtilizationDecrease := 0;
  List := null;
  for Task := 1 to n do Enlist(Task, List);
  while (UsedBuffers < b) and (UtilizationDecrease < InitialUtilization − U) do
    begin {of the frame-enhancement loop}
      find the first Tuple in List such that Tuple.AdditionalFrames ≤ b − UsedBuffers;
      if Tuple = null
       then break; {leave the loop and notify that the system is unschedulable}
       else
        begin
         Task := Tuple.Task;
         ExtraFrames := Tuple.AdditionalFrames;
         EnhancedFrameSize[Task] := EnhancedFrameSize[Task] + ExtraFrames;
         UsedBuffers := UsedBuffers + ExtraFrames;
        UtilizationDecrease := UtilizationDecrease+ExtraFrames·Tuple.RelativeDecrease;
         remove from List any Tuple such that Tuple.Task = Task;
         Enlist(Task, List);
        end;
    end; {of the frame-enhancement loop}
  if (UtilizationDecrease < InitialUtilization − U)
    then return("The task system is unschedulable")
    else return("The task system is schedulable", EnhancedFrameSize[1...n]);
end.
```

The complexity of Algorithm 1 is $O(nb^2(n+b))$.

After each iteration of the frame-enhancement loop in Algorithm 1, the achieved total utilization reduction would be the largest if the number of buffers would be exactly $2 \cdot$**UsedBuffers**. This statement can be proven by transforming an arbitrary set of n enhanced frames consisting of **UsedBuffers** original frames to match the configuration specified in **EnhancedFrameSize**$[1 \ldots n]$ without increasing the maximum utilization factor.

When Algorithm 1 claims that a task system is unschedulable, two cases can occur: **List** is empty, or there is no more buffers available. In the first case, the utilization factor is larger than the utilization bound U even after the best possible sizes were chosen for the enhanced frames. Thus, there is no solution for Problem 1 indeed. For the second case, let **Tuple** be the first tuple in **List**. Assume that the number of buffers equals to $2b'$ where $b' =$ **UsedBuffers**+**Tuple.AdditionalFrames**. The maximum utilization reduction is achieved by adding **Tuple.AdditionalFrames** into **EnhancedFrameSize**[**Tuple.Task**]. The new utilization $U' =$ **InitialUtilization** $-$ (**UtilizationDecrease** $+$ **Tuple.AdditionalFrames** $\cdot$ **Tuple.RelativeDecrease**) is no larger than the least utilization achievable with $2b$ buffers. Hence, if $U' > U$, there is no solution for Problem 1. If $U' \leq U$ and Problem 1 has a solution, the utilization factor for that solution $U_{solution}$ is at least U'. In other words, $U' \leq U_{solution} \leq U \leq$ **InitialUtilization** $-$ **UtilizationDecrease**. This means that if Algorithm 1 claims that a task system is unschedulable, it is very likely that there is no solution for the problem. When such a solution exists, its maximum utilization factor lies between U' and U.

3 Multiframe Task Model

The previous section introduces the task buffering technique. When applied to an irregular task system, this method can characterize the execution requirements more precisely by using the Enhanced Frame Model and, therefore, can achieve better resource utilization. Unfortunately, task buffering relies on postponement of task deadlines. When current delay bounds may not be extended, this method is not able to yield considerable improvement. This feature restricts applicability of the task buffering technique. Besides, this solution improves utilization of one resource (processing power, network link bandwidth) by essentially decreasing utilization of another one (buffer space). Even though memory is not as scarce as processing power, wasting of buffer space is not a desirable side effect. The approach we present below is free from this flaw and is based on the fact that large frames of a task occur intermittently with smaller frames located in between. For instance, video streams encoded in the MPEG standard consist from big I-frames followed by a number of smaller B-frames and P-frames. By taking this internal structure of tasks into account explicitly, one can derive higher utilization bounds. It allows us to recognize schedulability of many systems which were considered unschedulable when the tasks were described either by the regular periodic model or by the EFM.

[11] introduces a *MultiFrame task Model (MFM)*, studies this model when fixed-priority scheduling is

employed, and obtains a much better utilization bound than the testing bound for the regular periodic model. The major results of that paper are summarized below. Please see [11] for details.

Definition 3 *A* **multiframe real-time task** *is a tuple* (Γ, P) *where* Γ *is an array* $(C^0, C^1, \ldots, C^{N-1})$ *of* $N \geq 1$ *execution times, and* P *is the minimum separation time, i.e. the ready times of two consecutive frames must be at least* P *time units apart. The execution time of the* i*-th frame of the task is* $C^{((i-1) \bmod N)}$ *where* $i \geq 1$*. The deadline of each frame is* P *time units after its ready time.*

For example, $T_1 = ((3,2),4)$ is a multiframe task with the minimum separation time of 4. Odd frames have the execution time of 3 while execution of an even frame requires 2 time units.

Unlike traditional task models which assign the maximum execution time to each frame, the MFM takes into account fluctuating execution requirements of multimedia tasks. As a result, more multimedia tasks can be scheduled and higher resource utilization is achieved.

Below we define what it means for a multiframe task to be *accumulatively monotonic*:

Definition 4 *An array* $(C^0, C^1, \ldots, C^{N-1})$ *of execution times is said to be* **accumulatively monotonic (AM)** *if* $\exists\ m \in (0 : N-1)$ *such that* $\forall\ i, j \in (0 : N-1)$: $\Sigma_{k=m}^{m+j} C^{(k \bmod N)} \geq \Sigma_{k=i}^{i+j} C^{(k \bmod N)}$. *A multiframe task* $T = ((C^0, C^1, \ldots, C^{N-1}), P)$ *is said to be* **AM** *if its array of execution times is AM. A frame with execution time* C^m *is called a* **peak frame** *of* T*.*

Since every multiframe task can be represented as an AM multiframe task, we assume that all considered tasks satisfy the AM property. Without loss of generality, we also assume that the first frame of each multimedia task is its peak frame.

The following definition introduces the notion of *irregularity factors* of a multiframe task and a multiframe task system:

Definition 5 *Let* $S = \{T_i = ((C_i^0, C_i^1, \ldots, C_i^{N_i-1}), P_i) | 1 \leq i \leq n\}$ *be a multiframe task system. An* **irregularity factor** *of task* T_i *is defined to be* $r_i = \begin{cases} \frac{C_i^0}{C_i^1} & \text{if } N_i \geq 2 \\ 1 & \text{if } N_i = 1 \end{cases}$ $r = \min_{i=1}^{n} r_i$ *is called an* **irregularity factor** *of task system* S*.*

Let us consider fixed-priority scheduling of a multiframe task system. Because execution requirements of higher priority tasks are described more precisely, a multiframe task has a better chance to pass the critical instance test than an analogous regular task. This leads to a higher utilization bound:

Theorem 3 ([11]) *For fixed-priority scheduling of systems containing* n *multiframe tasks,* $r \cdot n \cdot ((\frac{r+1}{r})^{\frac{1}{n}} - 1)$ *can be used as a utilization bound*

If $r = 1$, the derived bound is the same as the famous $n(2^{\frac{1}{n}} - 1)$ bound for regular task systems. The larger the irregularity factor r of a multiframe task system is, the higher the utilization bound

name	*Max-I*	*Max-P*	*Max-B*	*Ave*	*Frame No.*	*Frame Pattern*
bike.mpg	116288	75752	26184	34270	150	IBBPBB
tennis.mpg	223320	167200	50672	66452	150	IBBPBB
mobile.mpg	165352	68112	44624	38336	150	IBBPBBPBBPBBPBB
canyon.mpg	26752	21120	7496	7936	1758	IBBPBB
jfk.mpg	65184	56392	32368	21905	156	IBBPBB
Red.mpg	211680	222504	143000	23933	1200	IBBBBBBBBBPBBBBB BBBBPBBBBBBBBB

Table 1: Characteristics of 6 MPEG video clips (frame sizes are in bits)

becomes. For instance, when $r = 3$ and systems consist of ten tasks, the bound is 22% higher than one for analogous regular task systems.

4 Experiments

This section quantitatively evaluates benefits of employing the proposed task models. Unless specified otherwise, the presented results are obtained in the context of rate-monotonic fixed-priority scheduling. We simulate real-time transmission of MPEG video streams over a network link with a fixed bandwidth (note that our experiments can be interpreted more generally as real-time processing of MPEG video streams in a system with some fixed processing ability). The video streams consist of periodic frames; each frame has to be sent before the arrival of the next frame. Every MPEG stream is composed of three types of frames (I-frame, P-frame, and B-frame) occurring with some repeating pattern. The size of an I-frame is usually bigger than ones of P-frames. I-frames and P-frames are much larger than B-frames. We use forty five different MPEG video clips in our experiments. Parameters of some of the streams are shown in Table 1.

Every video stream is represented as a task. When a stream is modeled as a regular task, the period and deadline of this task are the same as the interval between two successive frames of the stream; the execution requirement of the regular task is equal to the size of the maximum frame in the stream divided by the link bandwidth. When a stream is modeled as a multiframe task, the minimum separation time equals to the gap between two successive frames of the stream; the array Γ of execution times is of the same length as the repeating pattern of the stream; each execution time in Γ is equal to the maximum size of the corresponding frame in the repeating pattern divided by the bandwidth. Patterns of the considered video streams are such that all constructed multiframe tasks possess the AM property.

To solve the problem of the limited amount of frames in each stream, we repeat the clip over and over again to make the stream duration sufficiently long.

In general, our simulations are performed using the following three steps:

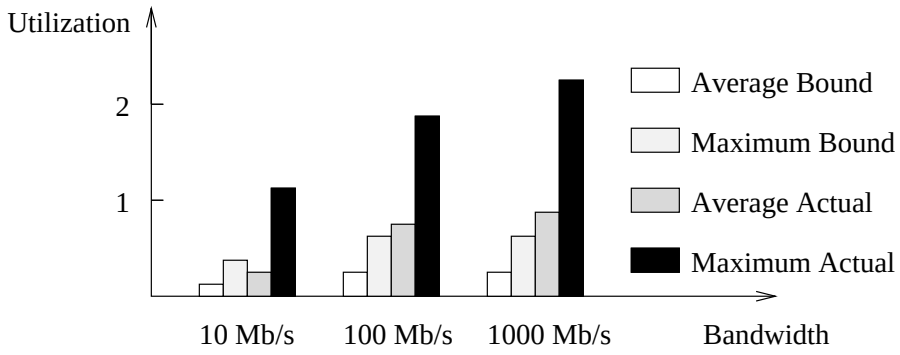

Figure 1: Resource utilization for the regular task model

1. Initially, there is no video streams to transmit.

2. A stream is randomly selected from the video pool, and its frame rate is randomly set to one of 24, 25, 30, 50, or 60 frames per second. Then the stream is added to the task system. Transmission of the streams is simulated for more than 10 minutes. If no deadline is missed, Step 2 is repeated.

3. The configuration of the task system as it was before adding the last stream is reported.

Each task system is simulated for at least 10 minutes (which corresponds to 18,000 frames of a video stream with the frame rate of 30 frames per second). This duration is long enough to detect overflow for most of overloaded systems. However, some overloaded systems can be considered as schedulable because they miss a frame deadline for the first time already after the simulation is stopped. Therefore, the reported maximum load can slightly exceed the actual allowed maximum load.

The average utilization factor U^{ave} of simulated task systems is calculated more conservatively than implied by its definition. Instead of considering all frames of a video stream, our calculation of U^{ave} for each stream is based on its repeating pattern and maximum sizes of I-frames, P-frames, and B-frames.

First, we compare actual utilization reported during the simulation with utilization achieved when the video streams are described using the regular task model and schedulability is tested based on the $n(2^{\frac{1}{n}} - 1)$ bound. Figure 1 shows that the actual average utilization factor (Average Actual) is much larger than the average utilization factor when the schedulability test is employed (Average Bound). The same is true for achievable maximum utilization: the actual maximum utilization factor (Maximum Actual) is much bigger than the maximum utilization factor reachable by the bound-based method (Maximum Bound). As the link bandwidth increases, achieved resource utilization grows. It happens because irregularities of individual video streams are smoothed out by combining a larger amount of streams into the task system. The main conclusion of this experiment is that using the regular task model and the $n(2^{\frac{1}{n}} - 1)$ bound leads to inefficient resource utilization.

The next experiment studies an impact of task buffering on resource utilization. When the maximum utilization factor is larger than the testing bound, additional buffers are introduced and Algorithm 1

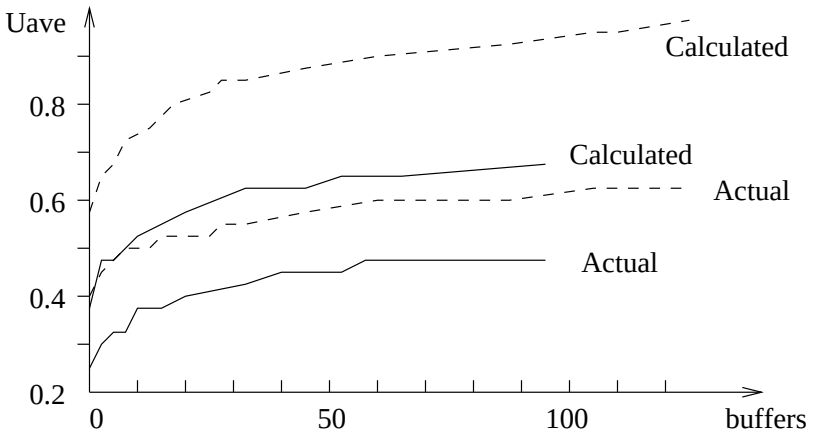

Figure 2: Impact of task buffering on the average utilization factor

is applied to transform the considered system of regular tasks to a system of EFM tasks. The number of frames in an enhanced frame is not allowed to exceed the length of the repeating pattern of the corresponding video stream. The execution time of each enhanced frame is calculated based on the repeating pattern and maximum execution times of I-frames, P-frames, and B-frames. The link bandwidth is assumed to be 100 Mb/s. Figure 2 shows achievable average utilization factors as functions of the amount of additional buffers. The solid lines represent the results for the rate-monotonic scheduler and $n(2^{1/n} - 1)$ testing bound. The dashed lines display utilization factors for the EDF scheduler and its testing bound equal to 1. For each scheduler, we provide two utilization factors: actual (reported by our simulations) and calculated (based on parameters of the enhanced frames). As one can see, calculated utilization approaches the corresponding testing bound when the number of additional buffers increases. The actual utilization factors fall below calculated one because of imprecise task characterization: many enhanced frames of a video stream are smaller than its maximum enhanced frame. The experiment demonstrates that the more buffers are added, the more video streams can be scheduled, i.e. the more beneficial the transformation to the EMF system becomes.

Figure 3 compares the maximum utilization factors for different task models. The regular model is used with association with $n(2^{\frac{1}{n}} - 1)$ bound. Deadlines for the ADM are selected to be 3 times as large as the periods of the corresponding tasks. The MFM uses $r = 4$ as the irregularity factor of task systems. The EFM limits the size of each enhanced frame to two original frames. The experiment shows that the EFM outperforms the other models and is able to provide the maximum utilization factor higher than 1.

Our last experiment investigates applying the task buffering technique to the MultiFrame Model (i.e. transforming MFM task systems to EFM systems). Once again, the link bandwidth is assumed to be 100 Mb/s, and sizes of enhanced frames are limited to two original frames. Figure 4 presents the maximum achieved utilization as a function of the irregularity factor of the task system. The numbers to

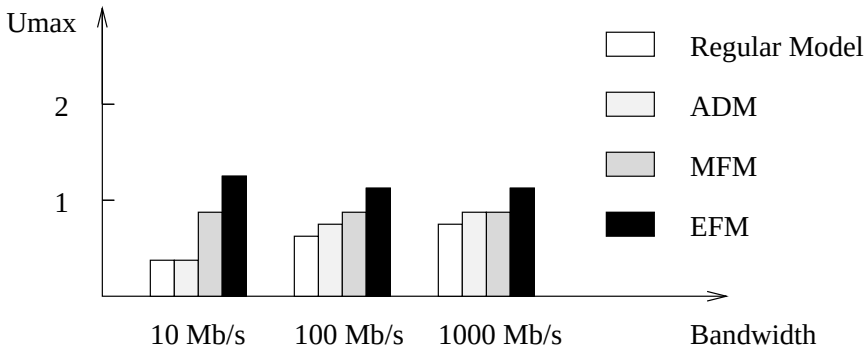

Figure 3: Comparison of the maximum utilization factors achieved by different task models

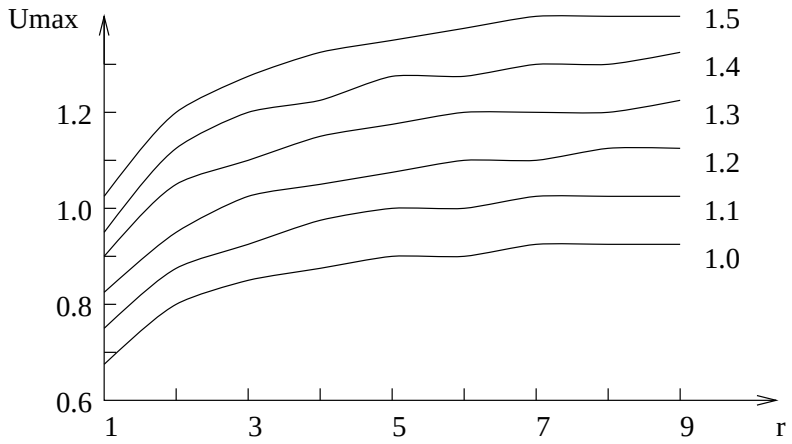

Figure 4: Combining the MFM and the EFM models

the right from each curve show ratios of the maximum utilization factor of original frames to the maximum utilization factor of enhanced frames. The curve with the ratio of 1.0 corresponds to data from Table 1 in [11]. This experiment demonstrates that combining the MFM (explicitly considering the internal structure of multimedia tasks) and the EFM (employing task buffering and deadline postponement) can drastically improve resource utilization.

Even though the proposed models allow us to achieve high maximum utilization factors $U^{max} > 1$ and considerably increase the average utilization, the latter certainly cannot exceed 100%, i.e. $U^{ave} \leq 1$.

5 Conclusion

Efficiency of bound-based schedulability tests makes them a very promising approach to be used for securing performance guarantees in emerging real-time multimedia applications. Unfortunately, most of the existing tests are derived for task models which assume constant execution time for every instance of a task. When applied to the multimedia domain, these models fail to characterize bursty multimedia

data precisely. As a result, employing these schedulability tests leads to inefficient resource utilization.

This paper presented a notion of irregular tasks which takes into account the fact that different instances of a task may need different execution times. Two modeling techniques — Enhanced Frame Model (EFM) and MultiFrame Model (MFM) — were discussed and combined for analyzing the schedulability of irregular tasks. A precise EFM characterization of a task system can be obtained by applying task buffering. This method can greatly improve resource utilization but is applicable only if some task deadlines can be postponed and additional buffer space is available. It was proven that the problem of transforming a periodic task system to an enhanced-frame task system is NP-complete. An effective heuristic polynomial-time algorithm for such transformations was derived.

The MFM model can be used for describing tasks which have a repeating pattern of execution requirements, e.g. for characterizing MPEG video streams. It was shown that employing the MFM can yield very high utilization bounds.

Experiments with real-time video transmission confirmed that using the proposed task models can drastically improve resource utilization in multimedia systems.

References

[1] J.-Y. Chung, J. W. S. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computer*, 39(9), 1990.

[2] D. L. Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[3] T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9(1), January 1995.

[4] S. Gorinsky, S. Baruah, T. Marlowe, and A. Stoyenko. Exact and efficient analysis of schedulability in fixed-packet networks: a generic approach. In *INFOCOM'97*, April 1997.

[5] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[6] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, December 1991.

[7] S. S. Lam, S. Chow, and D. K. Y. Yau. An algorithm for lossless smoothing of mpeg video. In *ACM SIGCOMM*, September 1994.

[8] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, December 1990.

[9] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1), January 1973.

[10] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD thesis, MIT, 1983.

[11] A. K. Mok and D. Chen. A multiframe model for real-time tasks. In *IEEE Real-Time Systems Symposium*, December 1996.

Appendix

A Definition of the Knapsack problem

Problem 3 (Knapsack) *Given a finite set U, a "size" $s(u) \in Z^+$ and a "value" $v(u) \in Z^+$ for each $u \in U$, and positive integers B and K, is there a subset $U' \subseteq U$ such that $\Sigma_{u \in U'} s(u) \leq B$ and such that $\Sigma_{u \in U'} v(u) \geq K$? [5]*

B Proof of Theorem 1

Problem 1 is in NP since we can guess n integer numbers k_i, $1 \leq i \leq n$, and test whether $\Sigma_{i=1}^{n} k_i \leq b$ and $\Sigma_{i=1}^{n} (A_i^{k_i}/(k_i \cdot P_i)) \leq U$ in linear time of n.

Problem 1 is NP-hard. We can polynomially reduce the NP-complete Knapsack problem to Problem 1.

We transform each u from an instance of the Knapsack problem into an irregular task $T = (\Gamma, P, D)$ such that $D = P$, the $(s(u)+1)$-th frame of the task has execution time 1, all frames after both $(s(u)+1)$-th and B-th frames have execution time 0, and all other frames have execution time $C^{max} = 1 + (s(u)+1) \cdot v(u) \cdot P/M$ where M and P are as specified below. T is an irregular task. Its maximum utilization factor is C^{max}/P. Its average utilization equals to 0. Figure 5 illustrates execution times of T for three different cases. Note that there is only one way to reduce the maximum utilization factor of T: $(s(u)+1)$ frames have to be combined into one enhanced frame. This reduces the utilization factor by $C^{max}/P - (s(u) \cdot C^{max} + 1)/((s(u)+1) \cdot P) = (C^{max} - 1)/((s(u)+1) \cdot P) = v(u)/M$. For the resulting irregular task system, we choose $b = B + n$ and select M and P such that $U = \Sigma(C^{max}/P) - K/M$ and $C^{max}/P \leq 1$. For each task T, either $(s(u)+1)$ frames are combined into an enhanced frame or the enhanced frame remains to consist of one frame. Thus, k is either 1 or $s(u)+1$. Corresponding values of the arrival bound function are $A^1 = C^{max}$ and $A^{s(u)+1} = s(u) \cdot C^{max} + 1$.

If Problem 1 can be solved for the constructed irregular task system, we have a number k for each task T such that $\Sigma k \leq b$, and $\Sigma A^k/(k \cdot P) \leq U$. For each task T, k is either 1 or $s(u)+1$. It means that we have a set U' of u such that $\Sigma_{u \notin U'} 1 + \Sigma_{u \in U'} (s(u)+1) \leq b$, which implies $\Sigma_{u \in U'} s(u) \leq B$, and $\Sigma A^k/(k \cdot P) = \Sigma_{u \notin U'} C^{max}/P + \Sigma_{u \in U'} ((s(u) \cdot C^{max} + 1)/((s(u)+1) \cdot P)) \leq U$. This means $\Sigma_{u \notin U'} C^{max}/P + \Sigma_{u \in U'} (C^{max}/P - v(u)/M) \leq \Sigma(C^{max}/P) - K/M$. Therefore, we have $\Sigma_{u \in U'} (v(u)) \geq K$. This is exactly a solution for the original Knapsack problem.

Thus, Problem 1 is NP-complete. **QED**

C Proof of Theorem 2

This proof is similar to the proof of Theorem 1 and is done by reducing the Knapsack problem to Problem 2.

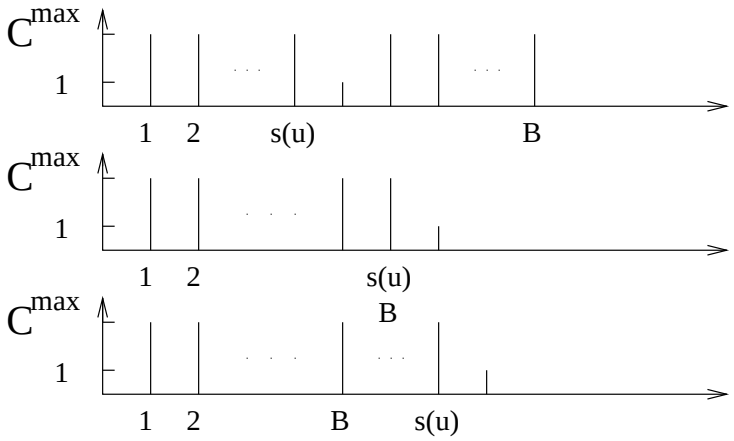

Figure 5: Three possible structures of irregular tasks

Problem 2 is in NP since we can guess n integer numbers k_i, $1 \leq i \leq n$, and test whether $\Sigma_{i=1}^{n} A_i^{k_i} \leq b$ and $\Sigma_{i=1}^{n}(A_i^{k_i}/(k_i \cdot P_i)) \leq U$ in linear time of n.

Problem 2 is NP-hard. We can polynomially reduce the NP-complete Knapsack problem to Problem 2.

We transform each u from an instance of the Knapsack problem to an irregular task $T = (\Gamma, P, D)$ such that $D = P$ and the task has only two kinds of execution times $C^{max} = 2P \cdot v(u)/M + s(u)$ and $C^{min} = s(u)$ where M and P are as specified below. The task switch between these two execution times frame by frame. T is an irregular task. Its maximum utilization factor is C^{max}/P. Its average utilization is $(C^{max} + C^{min})/(2P)$. There is only one way to reduce the maximum utilization factor of T: two frames have to be combined into one enhanced frame. This reduces the utilization factor by $C^{max}/P - (C^{max} + C^{min})/(2P) = v(u)/M$. For the constructed irregular task system, we choose $b = B + \Sigma C^{max}$ and select M and P such that $U = \Sigma(C^{max}/P) - K/M$ and $C^{max}/P \leq 1$. For each task T, either two frames are combined into an enhanced frame or the enhanced frame remains to consist of one frame. Thus, k equals to either 1 or 2. Corresponding values of the arrival bound function are $A^1 = C^{max}$ and $A^2 = C^{max} + C^{min}$.

If there is a solution for Problem 2, we have a set U' of u for which some corresponding tasks may have two-frame enhanced frames while the others have one-frame enhanced frames. We have: $\Sigma A^k = \Sigma_{u \notin U'} C^{max} + \Sigma_{u \in U'}(C^{max} + C^{min}) \leq b$ and $\Sigma(A^k/(k \cdot P)) = \Sigma_{u \notin U'} C^{max}/P + \Sigma_{u \in U'}((C^{max} + C^{min})/(2P)) \leq U$. The first inequality can be reduced to $\Sigma C^{max} + \Sigma_{u \in U'} C^{min} \leq B + \Sigma C^{max}$ and further to $\Sigma_{u \in U'} s(u) \leq B$. The second inequality can be reduced to $\Sigma(C^{max}/P) - \Sigma_{u \in U'}((C^{max} - C^{min})/(2P)) \leq \Sigma(C^{max}/P) - K/M$ and further to $-\Sigma_{u \in U'}(v(u)/M) \leq -K/M$ which can be written as $\Sigma_{u \in U'} v(u) \geq K$. This is exactly a solution for the original Knapsack problem.

Therefore, Problem 2 is NP-complete. **QED**