Technical Report

A preliminary idea for an 8-competitive, log2 DMAX + log2 log2 (1/U) asymptotic-space, interface generation algorithm for two-level hierarchical scheduling of constrained-deadline sporadic tasks on a uniprocessor

Björn Andersson

HURRAY-TR-110201

Version:

Date: 02-16-2011

A preliminary idea for an 8-competitive, log2 DMAX + log2 log2 (1/U) asymptotic-space, interface generation algorithm for two-level hierarchical scheduling of constrained-deadline sporadic tasks on a uniprocessor

Björn Andersson

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

Consider a single processor and a software system.The software system comprises components and interfaceswhere each component has an associated interface and eachcomponent comprises a set of constrained-deadline sporadictasks. A scheduling algorithm (called global scheduler) determinesat each instant which component is active. The activecomponent uses another scheduling algorithm (called localscheduler) to determine which task is selected for executionon the processor. The interface of a component makes certaininformation about a component visible to other components;the interfaces of all components are used for schedulabilityanalysis. We address the problem of generating an interfacefor a component based on the tasks inside the component. Wedesire to (i) incur only a small loss in schedulability analysisdue to the interface and (ii) ensure that the amount of space(counted in bits) of the interface is small; this is because suchan interface hides as much details of the component as possible.We present an algorithm for generating such an interface.

A preliminary idea for an 8-competitive, $\log_2 \mathrm{DMAX} + \log_2 \log_2 \frac{1}{\mathrm{U}}$ asymptotic-space, interface generation algorithm for two-level hierarchical scheduling of constrained-deadline sporadic tasks on a uniprocessor

Björn Andersson
CISTER Research Unit, ISEP/IPP
Polytechnic Institute of Porto
Porto, Portugal
bandersson@dei.isep.ipp.pt

***Abstract*—Consider a single processor and a software system. The software system comprises components and interfaces where each component has an associated interface and each component comprises a set of constrained-deadline sporadic tasks. A scheduling algorithm (called global scheduler) determines at each instant which component is active. The active component uses another scheduling algorithm (called local scheduler) to determine which task is selected for execution on the processor. The interface of a component makes certain information about a component visible to other components; the interfaces of all components are used for schedulability analysis. We address the problem of generating an interface for a component based on the tasks inside the component. We desire to (i) incur only a small loss in schedulability analysis due to the interface and (ii) ensure that the amount of space (counted in bits) of the interface is small; this is because such an interface hides as much details of the component as possible. We present an algorithm for generating such an interface.**

I. Introduction

Software design for embedded computer systems is affected by the steadily increasing (i) supply of execution-speed of microprocessors and (ii) demand from end-users for new features and improved application-level performance. Since these two factors increase each year, the complexity of embedded software (in terms of number of lines of code, function points or use cases) has now reached all-time-high levels. One way to deal with this complexity is to subdivide the software into *components* where each component has an interface (i) which is less complex than its corresponding component and (ii) which describes how it interacts or can interact with other components.

The problem of decomposing a (future) software system into components is typically driven by requirements on the system. Clearly functional requirements impact how the decomposition is done but non-functional requirements (also called quality-attributes or parafunctional requirements) play an important role as well. For example, a requirement may be that two different development teams (with their distinct expertise) should be able to work only on software that is within their expertise. Another requirement may be that the decomposition should be done so that already available (COTS) components can be used. Yet another requirement may be that two different functionalities should belong to different components because one functionality should not be able to obtain information about another component (confidentiality). Furthermore, in order to reduce overall certification cost (and re-certification cost in the event of design changes), it may be desirable for an architecture to have for each component, functionalities with no more than one criticality level. The problem of decomposing a software system into components is a significant problem in the discipline of software engineering (see [4] for an excellent coverage) but it is not the problem addressed in this paper. Therefore, we assume that the decomposition has already been done.

Typically, a software designer or developing organization or prime contractor (i) develops or acquires the needed components according to the decomposition mentioned above and then (ii) verifies correctness properties of the component assembly. For some correctness properties (typically logical correctness), it holds that the property is dependent only on a single component. This is ideal because it considerably simplifies integration of components into a working system. For many other correctness properties, however, the correctness property depends on more than one component. An example of this is real-time requirements. The response time of a task in a component depends on how much other tasks (for example higher-priority tasks) execute and these other tasks may be part of other components. In order determine if such a correctness property is true, each component must provide an interface which makes some information about the internals of the component visible to other components.

From systems integration perspective, the interface of a component should make as little as possible of the internals of the component visible to other components. (We can measure the ‘size’ of an interface as in how many bits are needed for storing the information that is made visible.) On the other hand, the more an interface makes visible to other components, the more accurate information is available to schedulability analysis techniques (or other quality-attribute analyses) and this reduces pessimism which can be translated into benefits such as (i) lower costs of hardware and/or (ii) lower power 'consumption'. Therefore, we must strike a suitable balance between schedulability and information hiding and in order to do so, we must quantify these.

The real-time research literature has provided a wealth of literature on the design of interfaces (see for example [8], [9], [7], [5]). The idea is to let each component be characterized by two numbers; typically (i) a bandwidth-like metric describing the fraction of the processor that the component may use and (ii) a period-like metric describing the granularity of this distribution of the used processor capacity. Some interfaces also allow a third number a, deadline, which can be used for a component to describe with slightly better accuracy how its requested processing capacity must be distributed. These interfaces have the benefits that (i) they are easy understand, (ii) they have associated algorithms for schedulability analysis and run-time scheduling and these allow practical issues such as non-processing resources to be shared between tasks in different components, (iii) they allow different local scheduling algorithms in different components and (iv) if all tasks are of the type implicit-deadline periodic or implicit-deadline sporadic then the loss of schedulability is typically small. Unfortunately, these interfaces can cause very poor performance for constrained-deadline sporadic tasks [1, page 3]. Specifically, there exist a task set which is schedulable with preemptive EDF on a single processor but with these interfaces, deadlines cannot be guaranteed although a processor k times as fast is used; and this holds for every finite value of k [1, page 3].

Constrained-deadline sporadic tasks are common in practice. For this reason, it behooves us to design new interfaces for such tasks. These interfaces should not suffer from the drawback of requiring an infinite speed processor when a speed-1 processor can do to meet deadlines. Also, the interfaces should make as little of the internals of the task sets in a component visible to other components.

Recent advances in interface design [1] has shown that even for constrained-deadline sporadic tasks, it is possible to create an interface where the loss in schedulability is provably small (can schedule every task set if provided a processor 8 times as fast). Unfortunately, it required $((\log_2 n)+1)\cdot((\log_2 \mathrm{TMAX})+2)\cdot((\log_2 \mathrm{DMAX})+2)\cdot((\log_2 \mathrm{CMAX})+1)$ bits; it is desired to reduce this space.

Therefore, in this paper, we present a new interface generation algorithm for constrained-deadline sporadic tasks to be scheduled on a single processor. We consider preemptive EDF to be used to as a scheduler in each component (local scheduler) and preemptive EDF to be used to schedule components upon the processor (global scheduler). The new interface generation algorithm offers two salient features: (i) for each task set which is feasible on a single 1-speed processor, it holds that if these tasks would be in components and scheduled on an 8-speed processor and the new interface generation algorithm is used then an offline schedulability test can guarantee that all deadlines are met and (ii) the space required by the interface of a component is asymptotically $\log_2 \mathrm{DMAX} + \log_2 \log_2 \frac{1}{\mathrm{U}}$ where DMAX is the maximum relative deadline of all tasks in the component and U is the sum of the utilization of all tasks in the component.

The remainder of this paper is organized as follows. Section II presents the system model and assumptions we make. Section III presents some results we will use. Section IV presents the new algorithm. Section V gives conclusions.

II. System model

We consider a system with the following assumptions:

A1. The system has a single processor;

A2. The software is composed of a set $\mathrm{COMP}=\{\mathrm{COMP}^1, \mathrm{COMP}^2, \ldots, \mathrm{COMP}^{\mathrm{K}}\}$ of K components;

A3. A component $\mathrm{COMP}^{\mathrm{k}}$ is composed of a set $\tau^k=\{\tau_1^k, \tau_2^k, \tau_3^k, \ldots, \tau_{n^k}^k\}$ of n^k constrained-deadline sporadic tasks (note that in this way, we restrict our attention to a 2-level hierarchy);

A4. A constrained-deadline sporadic task τ_i^k is characterized by the parameters T_i^k, C_i^k, D_i^k with the interpretation that τ_i^k releases a (possibly infinite) sequence of jobs with at least T_i^k time units between successive jobs of task τ_i^k and each job of τ_i^k requires C_i^k units of execution to be performed at least D_i^k time units after the release of the job. It is assumed that the release times of jobs cannot be controlled by the scheduling algorithm;

A5. A task which executes for L time units on a processor of speed S completes $L \cdot S$ units of execution;

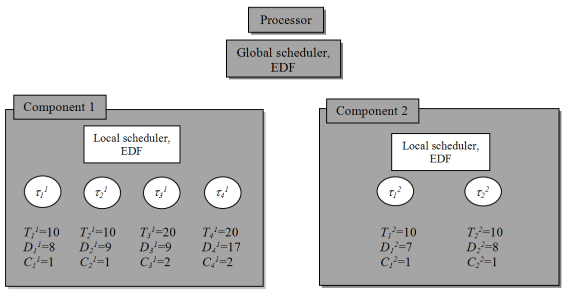
A6. If the speed of a processor is not explicitly specified, it is assumed that the speed is one;

A7. The parameters T_i^k, C_i^k, D_i^k are integers (we use the assumption of integer parameters only because it simplifies our discussion about the amount of storage needed for task sets and interfaces); arrivals of tasks are allowed to occur at non-integer times and preemptions are allowed to occur at non-integer times as well;

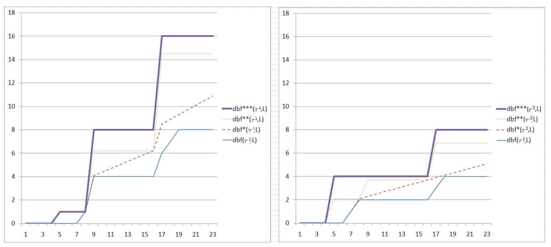
A8. A task needs no other resource than a processor;

A9. A component $\mathrm{COMP}^{\mathrm{k}}$ has a static interface $\mathrm{STATIC\_INTERFACE}^{\mathrm{k}}$ and a dynamic interface $\mathrm{DYNAMIC\_INTERFACE}^{\mathrm{k}}$. The static interface is composed of variables that remain constant over time (for example, pre-specified bandwidth) whereas the dynamic interface is composed of variables that may change with time (for example, a variable indicating whether there is any task in the component with unfinished execution at current time);

A10. There is a global scheduler GLOBAL_SCHED which decides at run-time, at every instant, which


Processor
Global scheduler, EDF
Component 1
Local scheduler, EDF
τ_1^1
τ_2^1
τ_3^1
τ_4^1
T_1^1=10 D_1^1=8 C_1^1=1
T_2^1=10 D_2^1=9 C_2^1=1
T_3^1=20 D_3^1=9 C_3^1=2
T_4^1=20 D_4^1=17 C_4^1=2
Component 2
Local scheduler, EDF
τ_1^2
τ_2^2
T_1^2=10 D_1^2=7 C_1^2=1
T_2^2=10 D_2^2=8 C_2^2=1

(a) A small example. (The interfaces and the schedulability tests are not shown.)



(b) Illustrations of dbf, $\mathrm{dbf}*$, $\mathrm{dbf}**$ and $\mathrm{dbf}***$ for the tasks in Figure 1(a).

Figure 1: A small example of the type of system we consider and also some important concept we will use.

component is assigned the processor. In every component COMP^k, there is a local scheduler $\mathrm{LOCAL\_SCHED}^\mathrm{k}$;

A11. The global scheduler takes decisions based on both the static and the dynamic interface of all components;

A12. The global scheduler is EDF [6];

A13. The local scheduler of COMP^k executes a task in τ^k at time t if the global scheduler has assigned the processor to COMP^k at time t;

A14. The local scheduler in a component takes decisions based only on the properties of the tasks in the component;

A15. The local scheduler in each component is EDF [6];

A16. There is a schedulability test for the global scheduler and a schedulability test for each local scheduler; these schedulability tests are used before run-time;

A18. The schedulability test for the global scheduler does not know tasks in each component and it does not know the dynamic interface of components. The schedulability test for the global scheduler takes as input the static interface of all components and outputs ‘schedulable’ or ‘unschedulable’;

A19. The schedulability test for the local scheduler of COMP^k only knows the tasks in τ^k and the interface of COMP^k. The schedulability test for the local scheduler COMP^k takes as input the tasks in τ^k and the interface of COMP^k and outputs ‘schedulable’ or ‘unschedulable’.

A20. There is an interface generation algorithm. This algorithm takes as input all tasks in one component and generates the static interface for this component. The interface should be generated so that for each component, the local schedulability test outputs ‘schedulable’. For this reason, it follows that once interfaces have been generated, determining if all tasks meet their deadline amounts to performing only the global schedulability test.

Figure 1(a) shows an example of such a system.

Recall that we address the problem of deciding which parameters should be used to represent the interfaces and how to select parameters for them and how the dynamic interface should be used at run-time. We are interested in doing so and fulfilling the following two (often conflicting) requirements:

R1. The loss in schedulability should be small;

R2. The interface of a component should reveal as little as possible about the tasks in the component.

For the purpose of our discussion, we need to quantify how well an interface fulfills the two requirements above. Therefore, we will define the concepts *competitive ratio* and *”narrowness”*. The former is related to R1 and the latter is related to R2.

We say that an interface generation algorithm A has competitive ratio R if R is the smallest number such that it holds that for every constrained-deadline sporadic task set partitioned into components, that if this task set can be scheduled on a single processor with EDF directly on the processor (that is, without components and without a global scheduler) then this task set can be guaranteed to meet its deadlines as well with interface generation algorithm A provided that the processor is R times as fast. Clearly, a low competitive ratio is desired. R=1 is the best one can get. R=∞ suggests that we pay a high price for compositionality.

In order to characterize the ”narrowness” of the interface, we consider the amount of storage needed to describe the interface.

III. Results we will use

A. Scheduling theory

1) Previously known results: The demand-bound function is a common concept for performing schedulability analysis [3]. The demand-bound function of a task τ_i^k is defined as:

$$\mathrm{dbf}(\tau_i^k, L) = \max(0, \left\lfloor \frac{L - D_i^k}{T_i^k} \right\rfloor + 1) \cdot C_i^k \tag{1}$$

Since we consider constrained-deadline sporadic tasks (for which $D_i^k \leq T_i^k$) we can rewrite it as:

$$\mathrm{dbf}(\tau_i^k, L) = \left\lfloor \frac{L + T_i^k - D_i^k}{T_i^k} \right\rfloor \cdot C_i^k \tag{2}$$

We can also define an upper bound on dbf of a task τ_i^k as:

$$\mathrm{dbf}{*}(\tau_i^k, L) = \begin{cases} 0 & \text{if } \mathrm{L} < \mathrm{D_i^k} \\ C_i^k + (L - D_i^k) \cdot \frac{C_i^k}{T_i^k} & \text{if } \mathrm{L} \geq \mathrm{D_i^k} \end{cases} \tag{3}$$

It has been shown in previous research (see for example Equation 3 in [2]) that

$$\mathrm{dbf}(\tau_i^k, L) \leq \mathrm{dbf}{*}(\tau_i^k, L) \leq 2 \cdot \mathrm{dbf}(\tau_i^k, L) \tag{4}$$

We can define these concepts also for tasks in a component k. Hence we get:

$$\mathrm{dbf}(\tau^k, L) = \sum_{\tau_j^k \in \tau^k} \left\lfloor \frac{L + T_j^k - D_j^k}{T_j^k} \right\rfloor \cdot C_j^k \tag{5}$$

$$\mathrm{dbf}{*}(\tau^k, L) = \sum_{\tau_j^k \in \tau^k} \mathrm{dbf}{*}(\tau_j^k, L) \tag{6}$$

$$\mathrm{dbf}(\tau^k, L) \leq \mathrm{dbf}{*}(\tau^k, L) \leq 2 \cdot \mathrm{dbf}(\tau^k, L) \tag{7}$$

Figure 1(b) illustrates these concepts.

Let us also define $\mathrm{U^k}$ as:

$$\mathrm{U^k} = \sum_{\tau_j^k \in \tau^k} \frac{C_j^k}{T_j^k} \tag{8}$$

We also define $\mathrm{DMAX^k}$ as:

$$\mathrm{DMAX^k} = \max_{\tau_j^k \in \tau^k} D_j^k \tag{9}$$

and

$$\mathrm{DMAX} = \max_{k \in \{1,2,\ldots,K\}} \mathrm{DMAX^k} \tag{10}$$

We let τ denote the union of tasks in all components. We can clearly define the functions dbf and dbf$*$ for τ as well.

We can use the concept of dbf$*$ to check schedulability — Lemma 1 shows this.

Lemma 1: If EDF is used as a local scheduler in each component and EDF is used as the global scheduler and $\sum_{k=1..K} U^k \leq 1$ and

$$\forall L \in \{1, 2, 3, \ldots, \mathrm{DMAX}\} : \sum_{k=1..K} \mathrm{dbf}{*}(\tau^k, L) \leq L \tag{11}$$

then all deadlines are met.

Proof: Follows from [3] and Equation 4. ■

2) New results: We say that an integer L is a two-power if L there is a non-negative integer l such that L can be written as $L = 2^l$. Clearly, 2 is a two-power; 4 is a two-power; 8 is a two-power. But also 0 and 1 are two-powers.

Recall that dbf$*$ is an upper bound on dbf. We will now define dbf $**$ which is an upper bound on dbf$*$. We will do so by obtaining the dbf$*$ at a value of L such that L is a two-power. Since dbf$*$ is monotonically increasing with L, it holds that the value obtained is also an upper bound on dbf$*$ for all values less than L. Formally, we define the following:

$$\mathrm{dbf}{**}(\tau^k, L) = \mathrm{dbf}{*}(\tau^k, 2^{\lceil \log_2 L \rceil}) \tag{12}$$

In addition, we define:

$$\mathrm{dbf}{***}(\tau^k, L) = \tag{13}$$
$$\begin{cases} 2^{\lceil \log_2 \mathrm{dbf}{**}(\tau^k, L) \rceil} & \text{if } \mathrm{dbf}{**}(\tau^k, \mathrm{L}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Let us also define $\mathrm{U{*}^k}$ as:

$$\mathrm{U{*}^k} = \begin{cases} \frac{1}{2^{\lfloor \log_2 \frac{1}{\mathrm{U^k}} \rfloor}} & \text{if } \mathrm{U^k} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Clearly, we have:

$$\mathrm{dbf}(\tau^k, L) \leq \mathrm{dbf}{*}(\tau^k, L)$$
$$\leq \mathrm{dbf}{**}(\tau^k, L) \leq \mathrm{dbf}{***}(\tau^k, L) \tag{14}$$

Lemma 2: If EDF is used as a local scheduler in each component and EDF is used as the global scheduler and $\sum_{k=1..K} U{*}^k \leq 1$ and

$$\forall L \in \{1, 2, 3, \ldots, \mathrm{DMAX}\} : \sum_{k=1..K} \mathrm{dbf}{***}(\tau^k, L) \leq L \tag{15}$$

then all deadlines are met.

Proof: Follows from Lemma 1 and the fact that $U^k \leq U{*}^k$ and $\mathrm{dbf}{*}(\tau^k, L) \leq \mathrm{dbf}{***}(\tau^k, L)$. ■

We will see later in this paper that the function $\mathrm{dbf}{***}(\tau^k, L)$ for component k can be represented in a compact form if $\mathrm{dbf}{***}(\tau^k, L)$ only needs to be evaluated for $L \in [1, 2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}]$. We should we aware that different components may have different maximum deadlines however so when we wish to perform schedulability analysis of a system comprising such components, we will need to evaluate $\mathrm{dbf}{***}(\tau^k, L)$ for large values of L. For this reason, let us define the following:

$$\mathrm{dbf}{****}(\tau^k, L) =$$
$$\begin{cases} \mathrm{dbf}{***}(\tau^k, L) & \text{if } \mathrm{L} \leq 2^{\lceil \log_2 \mathrm{DMAX^k} \rceil} \\ \mathrm{dbf}{***}(\tau^k, L) + (L - 2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}) \cdot U{*}^k & \text{otherwise} \end{cases}$$

Clearly, we have:

$$\forall L \in [1..2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}) : \mathrm{dbf}{***}(\tau^k, L) \leq \mathrm{dbf}{****}(\tau^k, L)$$

and

$$\forall L \in [1..2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}) : \mathrm{dbf}**(\tau^k, L) \leq \mathrm{dbf}****(\tau^k, L)$$

Together this gives us:

$$\forall L \geq 1 : \mathrm{dbf}*(\tau^k, L) \leq \mathrm{dbf}****(\tau^k, L)$$

Lemma 3: If EDF is used as a local scheduler in each component and EDF is used as the global scheduler and $\sum_{k=1..K} U*^k \leq 1$ and

$$\forall L \in \{1, 2, 3, \ldots, 2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}\} : \sum_{k=1..K} \mathrm{dbf}****(\tau^k, L) \leq L$$

then all deadlines are met.

Proof: Follows from the reasoning above. ■

3) Discussion about representation: For illustrative purpose, let us show $\mathrm{dbf}***(\tau^1, L)$ and $\mathrm{dbf}***(\tau^2, L)$ in tabular form and show them only for those L which are two-powers (because the function changes only at those L). The upper part of Figure 2a shows this. We let α^k denote the number of such L-values in component k such that the L-value is at most $\mathrm{DMAX^k}$. In this case, $\alpha^1 = 6$ and $\alpha^2 = 4$.

In general, we obtain α^k as:

$$\alpha^k = \lceil \log_2 \mathrm{DMAX^k} \rceil + 1 \quad (16)$$

If a certain value of $\mathrm{dbf}***$ is zero then we can represent that with a zero. If a certain value of $\mathrm{dbf}***$ is non-zero however, then we can (since it is a two-power) represent it by the logarithm of the value and then add one. This gives us, for each component, a sequence which characterizes $\mathrm{dbf}***$ of the component. The lower part of Figure 2a shows this. The length of the sequence is determined by α; component 1 has a sequence of length $\alpha^1 = 6$ whereas component 2 has a sequence of length $\alpha^2 = 4$.

Given that component 1 can be represented by a string of $\alpha_1 = 6$ numbers such that a number is at least zero and at most $\alpha_1 = 6$ and the numbers in the sequence are non-descending, let us consider all such possible sequences. Figure 2c shows this for $\alpha^1 = 6$ which is relevant for component 1. Figure 2d shows this for $\alpha^2 = 4$ which is relevant for component 2.

We can therefore represent $\mathrm{dbf}***$ for component k for values within $[1..\mathrm{DMAX^k}]$ with a single integer; we call it the sequence number of the component. For example, $\mathrm{dbf}***$ for component 1 for values within $[1..\mathrm{DMAX^k})$ can be represented by $\mathrm{sequence\_number}^1 = 44$. Also, $\mathrm{dbf}***$ for component 2 for values within $[1..\mathrm{DMAX^k}]$ can be represented by $\mathrm{sequence\_number}^2 = 3$.

Figure 2b shows how we can represent $U*^k$. We compute $U*^k$ from U^k. If U^k is zero then we can represent $U*^k$ with the number zero. If U^k is one then we can represent $U*^k$ with the number one. If U^k is half or smaller then we can represent $U*^k$ with the number $\lfloor \log_2 \frac{1}{U^k} \rfloor$+1. We let $\mathrm{util\_repr}^k$ denote the number representing $U*^k$.

Considering Lemma 3, we can represent a component k by <α^k, $\mathrm{sequence\_number}^k$, $\mathrm{util\_repr}^k$>. Figure 3 shows this representation. In order to know if this is an efficient representation however, we need to find an upper bound on $\mathrm{sequence\_number}^k$ and therefore, let us turn our attention to combinatorics.

B. Combinatorics

When proving the space required for our new interface later in this paper, we will need a result in combinatorics. This section proves that result in combinatorics.

Let us consider sequences of non-negative integers such that the elements in the sequence are in non-descending order. An example of such a sequence is <1,4,5>. Another example of such a sequence is <0,2,6,9,9,9,9,10>. Let $T(\alpha, \beta)$ denote the number of unique sequences of α elements such that the elements of the sequence is non-descending and the last element is at most β. It is assumed that α and β are positive integers. Figure 4 illustrates this.

Let us now reason about how to compute $T(\alpha, \beta)$. Let q denote the last number in the sequence. It is a number at most β and at least 0. We know that whatever number we pick, the remaining sequence would have a length $\alpha - 1$. Hence we obtain that:

$$T(\alpha, \beta) = \sum_{q=0}^{\beta} T(\alpha - 1, q) \quad (17)$$

This can be rewritten as:

$$T(\alpha, \beta) = (\sum_{q=0}^{\beta-1} T(\alpha - 1, q)) + T(\alpha - 1, \beta) \quad (18)$$

Observing that the left term on the right-hand side is equal to $T(\alpha, \beta - 1)$ gives us:

$$T(\alpha, \beta) = T(\alpha, \beta - 1) + T(\alpha - 1, \beta) \quad (19)$$

It also holds that:

$$\forall \beta \geq 1 : T(\alpha = 1, \beta) = \beta + 1 \quad (20)$$

and

$$\forall \alpha \geq 1 : T(\alpha, \beta = 1) = \alpha + 1 \quad (21)$$

Given these equations, we are now interested in finding an upper bound on $T(\alpha, \beta)$ as a closed-form expression.

Lemma 4:

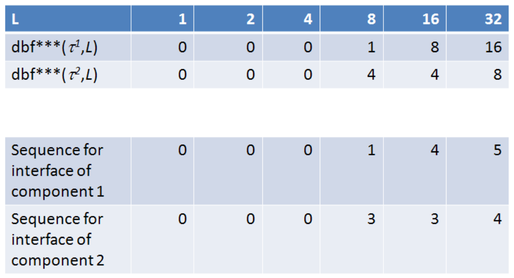
$$\forall \alpha \geq 1, \beta \geq 1 : T(\alpha, \beta) \leq 2^{\alpha+\beta} \quad (22)$$

Proof: Let us consider the claim:

$$\forall \alpha \geq 1, \beta \geq 1 : \alpha + \beta \leq l : T(\alpha, \beta) \leq 2^{\alpha+\beta} \quad (23)$$

where l is a positive integer.

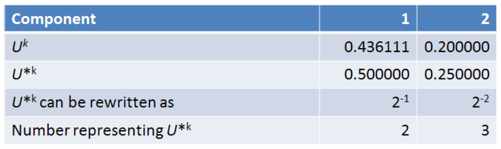
If we can prove Inequality 23 for each $l \geq 1$ then we know that Inequality 22 is true. We will prove Inequality 23 by using induction on l.



L	1	2	4	8	16	32
dbf***(τ^1,L)	0	0	0	1	8	16
dbf***(τ^2,L)	0	0	0	4	4	8

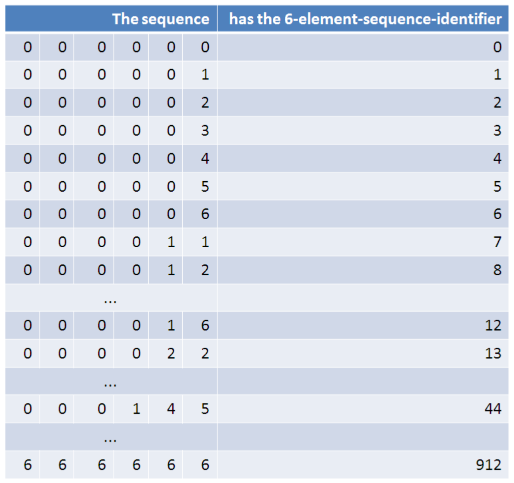
Sequence for interface of component 1	0	0	0	1	4	5
Sequence for interface of component 2	0	0	0	3	3	4

(a) Table of dbf $*$ $**(\tau^1, L)$ and dbf $*$ $**(\tau^2, L)$ and showing how sequences are generated.



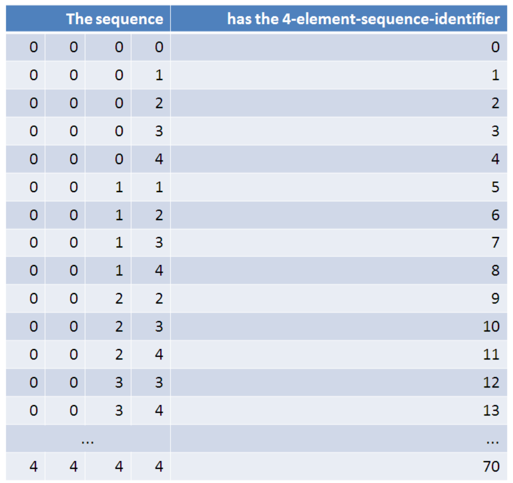
Component	1	2
U^k	0.436111	0.200000
U^{*k}	0.500000	0.250000
U^{*k} can be rewritten as	2^{-1}	2^{-2}
Number representing U^{*k}	2	3

(b) Interface of component 1 and component 2



The sequence						has the 6-element-sequence-identifier
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	0	2	2
0	0	0	0	0	3	3
0	0	0	0	0	4	4
0	0	0	0	0	5	5
0	0	0	0	0	6	6
0	0	0	0	1	1	7
0	0	0	0	1	2	8
…						
0	0	0	0	1	6	12
0	0	0	0	2	2	13
…						
0	0	0	1	4	5	44
…						
6	6	6	6	6	6	912

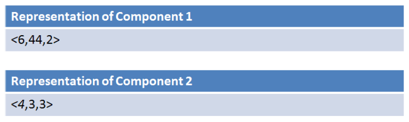
(c) Table showing the set of sequences for $\alpha^1 = 6$ and their 6-element-sequence number.



The sequence				has the 4-element-sequence-identifier
0	0	0	0	0
0	0	0	1	1
0	0	0	2	2
0	0	0	3	3
0	0	0	4	4
0	0	1	1	5
0	0	1	2	6
0	0	1	3	7
0	0	1	4	8
0	0	2	2	9
0	0	2	3	10
0	0	2	4	11
0	0	3	3	12
0	0	3	4	13
…				…
4	4	4	4	70

(d) Table showing the set of sequences for $\alpha^2 = 4$ and their 4-element-sequence number.

Figure 2: From dbf $*$ $**$ to sequence number and also how to represent utilization.



Representation of Component 1
<6,44,2>

Representation of Component 2
<4,3,3>

Figure 3: Interfaces for component 1 and component 2.

Base case: We claim: Inequality 23 is true for $l = 1$.

For this case we obtain that $\alpha = 1$ and $\beta = 1$ and using Inequality 20 gives us: $T(\alpha = 1, \beta = 1) = 2$. Hence, the base case is true.

Induction step: We claim: If Inequality 23 is true for $l = k$ then Inequality 23 is true for $l = k + 1$.

We prove the induction step by contradiction. Suppose that the induction step would be false. Then there is a positive integer k such that the following two inequalities are true:

$$\forall \alpha \geq 1, \beta \geq 1 \text{ such that } \alpha + \beta \leq k : T(\alpha, \beta) \leq 2^{\alpha+\beta} \quad (24)$$

and

$$\exists \alpha \geq 1, \beta \geq 1 \text{ such that } \alpha + \beta = k + 1 : T(\alpha, \beta) > 2^{\alpha+\beta} \quad (25)$$

Considering Inequalities 24 and 25, let α_0 and β_0 denote the values which exist in Inequality 25. This gives us:

$$T(\alpha_0 - 1, \beta_0) \leq 2^{\alpha_0 - 1 + \beta_0} \quad (26)$$

and

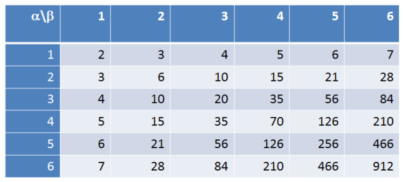
$$T(\alpha_0, \beta_0 - 1) \leq 2^{\alpha_0 + \beta_0 - 1} \quad (27)$$

and

$$T(\alpha_0, \beta_0) > 2^{\alpha_0 + \beta_0} \quad (28)$$

Applying Inequality 19 on α_0 and β_0 gives us:

$$T(\alpha_0, \beta_0) = T(\alpha_0, \beta_0 - 1) + T(\alpha_0 - 1, \beta_0) \quad (29)$$



α\β	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	6	10	15	21	28
3	4	10	20	35	56	84
4	5	15	35	70	126	210
5	6	21	56	126	256	466
6	7	28	84	210	466	912

Figure 4: Tabular specification of the function $T(\alpha,\beta)$.

Applying Inequality 29 on Inequalities 26, 27 and 28 yields:

$$2^{\alpha_0+\beta_0} < 2^{\alpha_0+\beta_0-1} + 2^{\alpha_0-1+\beta_0} \tag{30}$$

We can observe that the two terms on the right-hand side are the same. Hence, rewriting yields:

$$2^{\alpha_0+\beta_0} < 2^{\alpha_0+\beta_0-1} \cdot 2 \tag{31}$$

Further rewriting yields:

$$2^{\alpha_0+\beta_0} < 2^{\alpha_0+\beta_0} \tag{32}$$

This is a contradiction. Hence the induction step is true.

Since both the base case and the induction step are true, the induction argument yields Inequality 23 is true for each $l \geq 1$. Hence the lemma is true.

■

Lemma 5:

$$\forall \alpha \geq 1 : T(\alpha,\alpha) \leq 4^{\alpha} \tag{33}$$

Proof: Follows from Lemma 4. ■

C. Encoding and decoding sequences

Recall that we can represent a sequence as a sequence number. Figure 5a and 5b provides us with functions for doing this encoding/decoding.

IV. The new algorithm

Figure 6 shows pseudocode for generating the interface of a component. Let us now compute (asymptotically) the space needed (in bits) for the interface $< \alpha^k$, $\mathrm{sequence\_number}^k$, $\mathrm{util\_repr}^k >$. The space needed for α^k is (asymptotically) $\lceil \log_2 \alpha^k \rceil$ and using the expression for α^k gives us that the storage for α^k is $\lceil \log_2(\log_2 \mathrm{DMAX^k}) \rceil$.

The space needed for $\mathrm{sequence\_number}^k$ is asymptotically $\lceil \log_2(\mathrm{sequence\_number}^k) \rceil$. Using Lemma 5 gives us that the space needed for $\mathrm{sequence\_number}^k$ is asymptotically at most $\lceil 2 \cdot \log_2(2 \cdot \mathrm{DMAX^k}) \rceil$.

Let us now discuss the space needed for util_repr. If $U^k = 0$ then $U{*}^k = 0$. This gives us $\mathrm{util\_repr} = 0$ which requires just a single bit. If $U^k > 1/2$ then $U{*}^k = 1$. This gives us $\mathrm{util\_repr} = 1$ which requirest just a single bit as well. If $0 < U^k \leq 1/2$ then $U{*}^k = \frac{1}{2^{\lfloor \log_2 \frac{1}{\mathrm{U^k}} \rfloor}}$. This gives us the number $\mathrm{util\_repr} = \lfloor \log_2 \frac{1}{\mathrm{U^k}} \rfloor$. This can be stored with asymptotically with $\lceil \log_2 \lfloor \log_2 \frac{1}{\mathrm{U^k}} \rfloor \rceil$ bits. We can use two bits to decide which of the three above cases is the case. Hence, util_repr requires asymptotically at most $\lceil \log_2 \lfloor \log_2 \frac{1}{\mathrm{U^k}} \rfloor \rceil$.

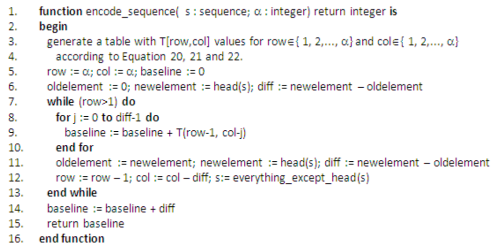
Putting all this together gives us that an upper bound on the space required for the interface of component k is asymptotically:

$$\log_2(\mathrm{DMAX^k}) + \log_2 \log_2 \frac{1}{\mathrm{U^k}} \tag{34}$$

Let us now reason about the competitive ratio. Let us consider a component k and let us consider a value L and compare it with $2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}$ and reason about the loss in terms of schedulability.

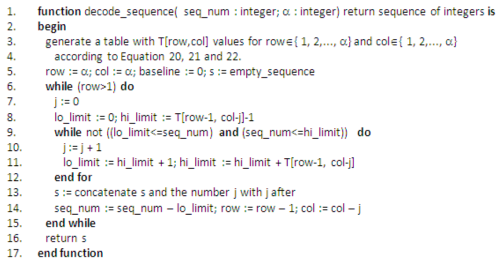
1) If $L \leq 2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}$ then we can reason as follows. The approximation of $\mathrm{dbf}{*}{*}{*}(\tau,L)$ by $\mathrm{dbf}{*}{*}{*}{*}(\tau,L)$ causes no loss. The approximation of $\mathrm{dbf}{*}{*}(\tau,L)$ by $\mathrm{dbf}{*}{*}{*}(\tau,L)$ causes a loss by a factor of two. The approximation of $\mathrm{dbf}{*}(\tau,L)$ by $\mathrm{dbf}{*}{*}(\tau,L)$ causes a loss by a factor of two. The approximation of $\mathrm{dbf}(\tau,L)$ by $\mathrm{dbf}{*}(\tau,L)$ causes a loss by a factor of two. Hence, we lose a factor of eight.
2) If $L > 2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}$ then we can reason as follows. When we compute $\mathrm{dbf}{*}{*}{*}{*}(\tau,L)$ we have two terms. One is $\mathrm{dbf}{*}{*}{*}(\tau,L)$ and the other is $(L - 2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}) \cdot U{*}^k$. Let us discuss the term $\mathrm{dbf}{*}{*}{*}(\tau,L)$. The approximation of $\mathrm{dbf}{*}{*}(\tau,L)$ by $\mathrm{dbf}{*}{*}{*}(\tau,L)$ causes a loss by a factor of two. The approximation of $\mathrm{dbf}{*}(\tau,L)$ by $\mathrm{dbf}{*}{*}(\tau,L)$ causes no loss. The approximation of $\mathrm{dbf}(\tau,L)$ by $\mathrm{dbf}{*}(\tau,L)$ causes a loss by a factor of two. Hence, we lose a factor of four. Let us discuss the term $(L - 2^{\lceil \log_2 \mathrm{DMAX^k} \rceil}) \cdot U{*}^k$. The approximation of U^k by $U{*}^k$ cause a loss by a factor of two. The loss in schedulability because of our approximation of $\mathrm{dbf}{*}{*}{*}{*}(\tau,L)$ is the maximum of the two terms. Hence, we lose a factor of eight.

Based on this reasoning, we can see that our new schedulability test which takes input from our interface generation



```
function encode_sequence( s : sequence; α : integer) return integer is
begin
  generate a table with T[row,col] values for row∈{ 1, 2,..., α} and col∈{ 1, 2,..., α}
    according to Equation 20, 21 and 22.
  row := α; col := α; baseline := 0
  oldelement := 0; newelement := head(s); diff := newelement – oldelement
  while (row>1) do
    for j := 0 to diff-1 do
      baseline := baseline + T(row-1, col-j)
    end for
    oldelement := newelement; newelement := head(s); diff := newelement – oldelement
    row := row – 1; col := col – diff; s:= everything_except_head(s)
  end while
  baseline := baseline + diff
  return baseline
end function
```

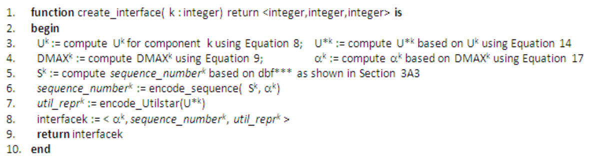
(a) Encoding.



```
function decode_sequence( seq_num : integer; α : integer) return sequence of integers is
begin
  generate a table with T[row,col] values for row∈{ 1, 2,..., α} and col∈{ 1, 2,..., α}
    according to Equation 20, 21 and 22.
  row := α; col := α; baseline := 0; s := empty_sequence
  while (row>1) do
    j := 0
    lo_limit := 0; hi_limit := T[row-1, col-j]-1
    while not ((lo_limit<=seq_num) and (seq_num<=hi_limit)) do
      j := j + 1
      lo_limit := hi_limit + 1; hi_limit := hi_limit + T[row-1, col-j]
    end for
    s := concatenate s and the number j with j after
    seq_num := seq_num – lo_limit; row := row – 1; col := col – j
  end while
  return s
end function
```

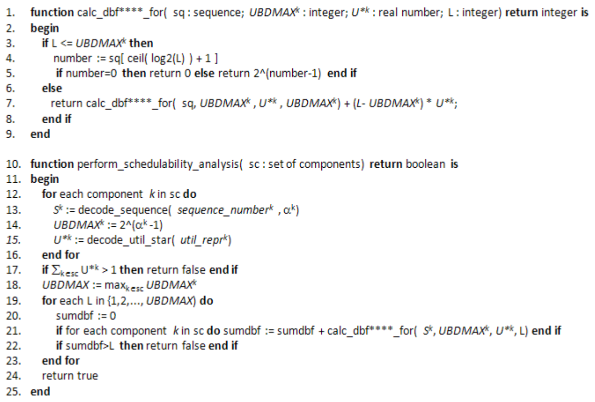
(b) Decoding

Figure 5: Encoding and decoding of sequence numbers.



```
function create_interface( k : integer) return <integer,integer,integer> is
begin
  $U^k$ := compute $U^k$ for component k using Equation 8;    $U^{*k}$ := compute $U^{*k}$ based on $U^k$ using Equation 14
  $DMAX^k$ := compute $DMAX^k$ using Equation 9;               $α^k$ := compute $α^k$ based on $DMAX^k$ using Equation 17
  $S^k$ := compute $sequence\_number^k$ based on dbf*** as shown in Section 3A3
  $sequence\_number^k$ := encode_sequence( $S^k$, $α^k$)
  $util\_repr^k$ := encode_Utilstar($U^{*k}$)
  interfacek := < $α^k$, $sequence\_number^k$, $util\_repr^k$ >
  return interfacek
end
```

(a) Creating an interface.



```
function calc_dbf****_for( sq : sequence; $UBDMAX^k$ : integer; $U^{*k}$ : real number; L : integer) return integer is
begin
  if L <= $UBDMAX^k$ then
    number := sq[ ceil( log2(L) ) + 1 ]
    if number=0 then return 0 else return 2^(number-1) end if
  else
    return calc_dbf****_for( sq, $UBDMAX^k$ , $U^{*k}$ , $UBDMAX^k$) + (L- $UBDMAX^k$) * $U^{*k}$;
  end if
end

function perform_schedulability_analysis( sc : set of components) return boolean is
begin
  for each component k in sc do
    $S^k$ := decode_sequence( $sequence\_number^k$ , $α^k$)
    $UBDMAX^k$ := 2^($α^k$-1)
    $U^{*k}$ := decode_util_star( $util\_repr^k$)
  end for
  if $\sum_{k \in sc} U^{*k} > 1$ then return false end if
  $UBDMAX := \max_{k \in sc} UBDMAX^k$
  for each L in {1,2,..., UBDMAX} do
    sumdbf := 0
    if for each component k in sc do sumdbf := sumdbf + calc_dbf****_for( $S^k$, $UBDMAX^k$, $U^{*k}$, L) end if
    if sumdbf>L then return false end if
  end for
  return true
end
```

(b) Performing schedulability analysis

Figure 6: Creating an interface and using interfaces for performing schedulability analysis.

algorithm gives us a competitive ratio of eight.

V. Conclusion

We have shown an 8-competitive, $\log_2 \mathrm{DMAX} + \log_2 \log_2 \frac{1}{\mathrm{U}}$ space, interface generation algorithm for constrained-deadline sporadic tasks on a single processor. We gave an informal argument why it is 8-competitive but left open the problem of proving it.

Acknowledgements

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência e a Tecnologia - FCT) and the European Commission through grant ArtistDesign ICT-NoE-214373 and Luso-American Development Foundation (FLAD).

References

[1] B. Andersson. A pseudo-medium-wide 8-competitive interface for two-level compositional real-time scheduling of constrained-deadline sporadic tasks on a uniprocessor. In *Proc. of 2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (Co-located with RTSS)*, 2009.

[2] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers*, 55(7):918–923, 2006.

[3] S. K. Baruah, A. K. Mok, and L. E. Rosier. Scheduling hard-real-time sporadic tasks on one processor. In *Proc. of 11th Real-Time Systems Symposium (RTSS)*, pages 182–190, 1990.

[4] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, second edition, 2003.

[5] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *Proc. of 28th Real-Time Systems Symposium (RTSS)*, pages 129–138, 2007.

[6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for the Computing Machinery*, 20:46–61, 1973.

[7] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *Proc. of 7th IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 75–84, 2001.

[8] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of 24th Real-Time Systems Symposium (RTSS)*, pages 2–10, 2003.

[9] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proc. of 25th Real-Time Systems Symposium (RTSS)*, pages 57–67, 2004.