Early Quantum Task Scheduling

P.G. Jansen, F. Hanssen, M. Lijding
Distributed and Embedded Systems group
Faculty of Computer Science, University of Twente
E-mail: {jansen,hanssen,lijding}@cs.utwente.nl

5th November 2002

Abstract

An *Early Quantum Task* (EQT) is a *Quantum EDF task* that has shrunk its first period into one quantum time slot. Its purpose is to be executed as soon as possible, without causing deadline overflow of other tasks. We will derive the conditions under which an EQT can be admitted and can have an immediate start. The advantage of scheduling EQTs is shown by its use in a buffered multi-media server. The EQT is associated with a multimedia stream and it will use its first invocation to fill the buffer, such that a client can start receiving data immediately[1].

Keywords real time scheduling, admission control, continuous media server, buffer management, jukebox server

1 Introduction

Early Quantum Scheduling is a real-time scheduling method that solves the problem of a too long waiting time for a task to enter the system. This waiting time occurs after the acknowledgement for the admission of a task, and is due to the regular scheduling rules. These rules normally determine that a task may execute when it has the highest priority. For instance, under *Earliest Deadline First* (EDF) this is the task with the earliest deadline. By breaking the priority rules temporarily the scheduler can speed up the start of a task considerably.

Consider for instance a multimedia client/server system, in which one or more media servers send media streams to consumers. Such a consumer may request a media stream and waits for the acknowledgement of the admittance. After the acknowledgement the server may then schedule the admitted stream amongst all the others, however the new stream has to await its regular turn for the first filling of its communication buffer. We can speed up the start of such a stream, by filling this buffer at the earliest time possible. This paper describes the condition under which such in-time buffer filling can be arranged, without endangering the deadlines of all other streams.

In the context of this paper we assume a single processor in which all tasks are non-preemptive and need a unit of computation time, and are started and ended at integer-time values. The computation units are called *synchronous quantum units* or *quanta*. The tasks are scheduled by EDF and we will denote this type of scheduling as *Quantum*

[1]This research is supported by the IBM Equinox program.

EDF (QEDF). QEDF scheduling simplifies the admission criteria of the newcomers to a simple algorithm. If the tasks are used to send or receive media, a task can be associated with a stream.

An *Early Quantum Tasks* (EQT) is derived from a normal QEDF task, however it executes its first invocation as soon as possible. We will illustrate the use of EQTs with a multimedia client/server. The advantage is a server cam start filling the buffer immediately.

EQTs could be used in any quantumised server. We use them in our jukebox multimedia server [11].

The structure of the paper is as follows. Section 2 refers to the related work. Section 3 introduces QEDF, section 4 describes EQTs and section 5 explains admission control of EQTs. Section 6 shows the use of EQTs in a multimedia server, explains how to avoid buffer over- and underflow. In section 7 the conclusions are drawn.

2 Related work

An EQT is constructed from a normal QEDF task in two steps. In the first step the QEDF task is transformed to an intermediate form, the *Push In Task* (PIT), and in the second step the PIT is transformed to an EQT. The PIT may execute its invocations always immediately and does not have to await its EDF turn. This PIT can be used as the ideal aperiodic server that may execute an aperiodic request immediately after having respected the inter-arrival times of the requests. It belongs to the class of *dynamic* servers as presented in [6]. These servers both work in the EDF domain of preemptable tasks and are called dynamic because they use EDF which is considered to embody a dynamic priority scheme. A PIT is close to a quantum task version of the *total bandwidth server*, or closer, to a quantum task version of the *improved total bandwidth server* from Spuri and Buttazzo [14, 15].

The total *bandwidth server* executes in a preemptable EDF environment. It is a task that serves aperiodic requests. When the k^{th} aperiodic request arrives at time $t = r_k$ it receives a deadline $d_k = \max(r_k, d_{k-1}) + C_k/U_s$, where C_k is the execution time of the request and U_s is the server utilisation. After assignment of this deadline the request is inserted into the "released queue" and scheduled as a normal EDF invocation.

The *improved total bandwidth server* shifts the actual execution to the earliest possible point in time, without endangering the deadlines of the other EDF tasks, thus giving the best service possible to the server. A disadvantage is, however, that this operation needs rather complex computation. We will show that our PIT does not need to do any computation and guarantees immediate execution without endangering the other deadlines. Furthermore we will exploit our PIT to speed up the start of QEDF tasks or streams.

Quantum tasks have been proposed earlier. Baruah et al. [3] introduced a theory related to quantum tasks called *Pfair scheduling*. The scheduling algorithm was called PD and it can be used to schedule tasks in a parallel environment. Under Pfair conditions a uniform execution rate is guaranteed by breaking tasks into quantum-length sub-tasks. Each sub-task must be executed within a window of time slots, the last of which must be the deadline. Anderson et al. [2] continued to work on this idea. They developed fundamental properties inherent to Pfair scheduling and could simplify the priority conditions of PD.

Buffered client/server systems are widely used and certainly not new. However, scheduling continuous media streams in-

volves also the need to avoid buffer under- or overflow. Korst et al. [9] compared a number of disk scheduling algorithms that can be used in a multimedia server for sustaining multiple variable-bit-rate (VBR) data streams and paid attention to the buffer under- and overflow. We assume a constant bit rate for the streams. According to [1] and [5] this is acceptable for variable bit rates when buffering big chunks of data.

3 Early Quantum EDF tasks

A PIT is a promoted QEDF task, that executes its new invocations as soon as possible. We need a PIT as an intermediate step to transform a QEDF task into a EQT. To describe this transformation process we will proceed as follows. First we introduce an EDF task set Γ and describe its well-known properties. Then we rewrite these properties to apply for QEDF tasks. Next we select one of the tasks to become a PIT. Successively we show that this task can execute its periodic invocations immediately after their releases without endangering the deadlines of the other tasks.

Given a task set $\Gamma = \{\tau_1, \ldots, \tau_n\}$ of n tasks with, for each $1 \leq i \leq n$:

$$\tau_i = (D_i, T_i, C_i)$$

where

D_i: the relative deadline. For quantum tasks $D_i = T_i$ is a positive integer value.

T_i: the period, the duration between two successive invocations.

C_i: the maximum run-time τ_i takes to complete each invocation. For quantum tasks $C_i = 1$.

We order tasks by deadline with the shortest deadline first. Quantum tasks are *synchronous*, and events such as *release*, *activation*, *finalisation* (the end of an activation), and *deadline* occur at integer times.

The *utilisation*, as defined by Liu and Layland [12], determines the fraction of processor time spent in execution of a task set. It is defined as:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{1}$$

which can be rewritten for quantum tasks as:

$$U = \sum_{i=1}^{n} \frac{1}{T_i} \tag{2}$$

We adapted Liu and Layland's theorem to quantum scheduling as follows:

Theorem 1 (QEDF feasibility (L&L)) *Given a synchronous quantum task set Γ scheduled under QEDF. The set is feasible if and only if:*

$$U \leq 1 \tag{3}$$

Proof The original theorem holds for preemptive EDF tasks and not for non-preemptive tasks. The proof is based on the fact that there is no difference between a preemptive set of quantum tasks and a non-preemptive set of tasks if release times and deadlines are quantum units, expressed as integer values. By construction, exactly the same scheduling patterns occur under preemption as well as under non-preemption and therefore both schedules must be equivalent, which concludes the proof of the theorem.
□

Our target is to speed up the introduction of tasks. This will cost some of the free capacity of the processor, which can be determined by computing the difference between the *processor capacity* and the *processor demand* . The processor demand function $H(t)$, defined by

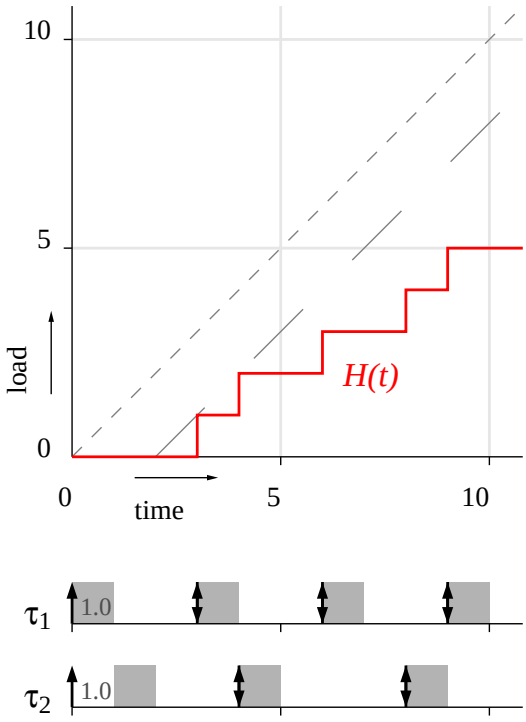


Figure 1: Capacity t and demand $H(t)$

Baruah [4] and Spuri [13] is suited to this purpose. $H(t)$ describes the amount of load that should be solved by the processor in a time interval t in order to meet all deadlines under EDF.

$$H(t) = \sum_{i=1}^{n} \left\lfloor \frac{t - D_i + T_i}{T_i} \right\rfloor C_i \qquad (4)$$

For quantum tasks this can be rewritten to:

$$H(t) = \sum_{i=1}^{n} \left\lfloor \frac{t}{T_i} \right\rfloor \qquad (5)$$

Figure 1 shows the processor capacity line under 45 degrees ($load = t$) and the processor demand function $H(t)$ for two QEDF tasks τ_1 and τ_2 with period 3 and 4 respectively. The task bars show the worst case schedule of τ_1 and τ_2.

Baruah [4] and Spuri [13], and later also Jeffay [8] rewrote the feasibility condition $U \leq 1$ for the original Liu and Layland theorem to:

Theorem 2 (QEDF feasibility (B&S)) *Given a synchronous quantum task set Γ scheduled under QEDF. The set is feasible if and only if:*

$$\forall t : 0 \leq t : H(t) \leq t \qquad (6)$$

It is not needed to go until the infinitum in order to verify this equation. There exist estimations for time lower-bounds[2] to conclude the correctness of equation 6. If it is possible to show the correctness until such a lower-bound, then the correctness for all $0 \leq t$ can be concluded.

The difference between the processor capacity t and the processor demand $H(t)$ is called slack. Slack can be interpreted as

[2]The Baruah bound or longest idle period (also called busy period [8]) can both be used as lower-bound

unused processor capacity. We define the *slack* $S(t)$ by:

$$S(t) = t - H(t) \tag{7}$$

The same arguments as those for the proof of theorem 1 can be used.

Note that $H(t)$ reflects the processor demand for any interval in time with length t and for any phase (shifted positions) between the EDF tasks. It is certainly not exclusively related to one experiment in which the tasks are all started simultaneously with a phase 0, as might be suggested by the task bars of figure 1.

Slack space can be used for additional activity and we will investigate the conditions under which this space can be used. A larger slack space allows for more successive new tasks to be introduced earlier. In particular we are interested in the greatest lower-bound S_{min} of all slack values. We can use this S_{min} for early introductions of tasks. S_{min} can be determined by inspecting those values for t at which deadlines expire and at which $H(t)$ is changed. This can be verified by inspection of equation 5. Consider for instance figure 1 where S_{min} is determined for τ_1 and τ_2. The graph shows the minimum slack line, running parallel to the processor capacity line. $S_{min} = 2$ is found at $t = 3$ and at $t = 4$.

In order to find the minimal value of the slack we have to inspect $S(t)$ for all deadlines until some time upper-bound. We expect that it is possible to determine a suitable time upper-bound, but we will not go into this procedure because of the computational costs. Instead we estimate the minimum value of $S(t)$ on the basis of the following theorem:

Theorem 3 (QEDF slack)

Given a synchronous quantum task set Γ, *with hyper-period* $\mathcal{T}$, *scheduled under QEDF, with a utilisation* $U \leq 1$, *processor demand* $H(t)$ *and slack* $S(t)$. *Then*

$$\begin{array}{l}\big(\forall t \in [T_{min}, \mathcal{T}) : S(t) > (1-U)t\big) \wedge \\ S(\mathcal{T}) = (1-U)\mathcal{T}\end{array} \tag{8}$$

Proof The proof has two parts. The first part determines $S(t)$ for $t \in [T_{min}, \mathcal{T})$ and the second part considers $S(t)$ for $t = \mathcal{T}$. First part:

$$\begin{array}{l}\forall t \in [1, \mathcal{T}) : \\ S(t) = t - H(t) = t - \sum_{i=1}^{n} \left\lfloor \frac{t}{T_i} \right\rfloor > \\ t - \sum_{i=1}^{n} \frac{t}{T_i} = t(1-U)\end{array}$$

which follows from (1) the definition of $H(t)$, (2) the property of the floor operator on the floating point values of the fraction t/T_i since $t < \mathcal{T}$, and (3) the definition of U. This completes the first part.

The second part is similar to the first part. Because $\mathcal{T}$ is the hyper-period $\mathcal{T}/T_i \in \mathbb{N}$.

$$\begin{array}{l}S(\mathcal{T}) = \mathcal{T} - H(\mathcal{T}) = \mathcal{T} - \sum_{i=1}^{n} \left\lfloor \frac{\mathcal{T}}{T_i} \right\rfloor = \\ \mathcal{T} - \sum_{i=1}^{n} \frac{\mathcal{T}}{T_i} = \mathcal{T}(1-U)\end{array}$$

which completes the second part.
□

The interpretation of this theorem is as follows. (1) If $U < 1$ then a slack $S(t) > (1-U)t$ is available for intervals of length t. (2) If $U = 1$ then all the intervals have at least a slack $S(t) > 0$, which implies $S(t) = 1$ (because integer values are involved), except for the case that *all* tasks have a deadline at hyper-period intervals $t = \mathcal{T}$. For such an interval there is exactly one execution for which S(t) must be 0.

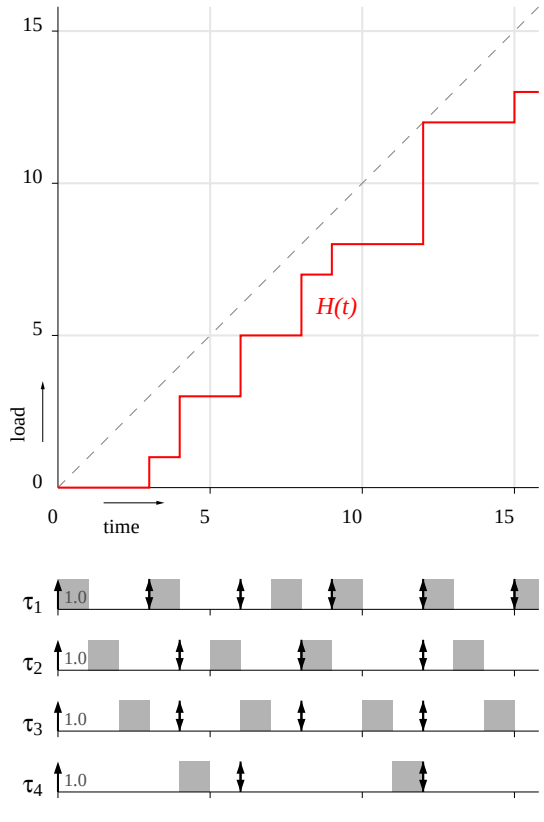


Figure 2: t and $H(t)$ of Γ_1

Example 1 *Given the specification of a set of QEDF tasks* Γ_1 *according to figure 2.*

The utilisation $U = 1$, so Γ_1 is feasible. According to theorem 3 $S_{min} > 0$, hence every execution has a slack of at least 1 quantum, except the last execution in a hyper-period interval $\mathcal{T} = 12$ at which point all tasks have an expiring deadline. τ_4 is the task that executes just before the hyper-period[3].
■

On the basis of theorem 3 we can estimate a slack lower-bound S_{est} with a simple expression instead of computing the greatest lower-bound S_{min} as stated in the following corollary.

Corollary 1 (Slack estimation)
Given a feasible QEDF set Γ *with utilisation* $U < 1$. *Let* T_{min} *be the smallest period of all tasks in* Γ *and* $S_{est} = \lceil (1-U)T_{min} \rceil$.

S_{est} *is a lower-bound for all quantum slack values in* Γ.

Proof From equation 7 and theorem 3 it follows that:

$$\begin{aligned} &\forall t \in [T_{min}, \mathcal{T}] : \\ &(1-U)T_{min} \leq (1-U)t < S(t) \implies \\ &S_{est} = \lceil (1-U)T_{min} \rceil \leq \lceil S(t) \rceil \end{aligned}$$

since for all expiring deadlines in $[T_{min}, \mathcal{T}]$, $S(t)$ must be an integer value, hence we may take the ceiling of $(1-U)T_{min}$ for the lower-bound of all values of S(t), which completes the proof.
□

[3]Any other choice in case of deadline ties could have been chosen. Note however that the choice in case of ties should be deterministic.

Example 2 (τ_1 and τ_2 of Γ_1)
Given the specification of a set of tasks as shown in figure 1.

According to corollary 1 the estimated minimum slack value is $\lceil (1-U)T_{min} \rceil = \lceil (1-7/12)3 \rceil = \lceil 15/12 \rceil = 2$, which is equal to the real minimum value. Note that for other cases this estimation might be smaller than the minimum value.
■

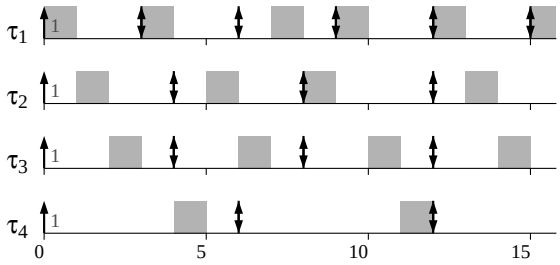
4 Early Quantum Tasks

We will construct EQTs in steps. In the first step we transform a newly arrived and admitted QEDF task τ_q to a PIT $F(\tau_q)$ by shifting all executions of τ_q to the first time slot after release. In the second step we transform $F(\tau_q)$ to an EQT $E(F(\tau_q))$ by splitting up $F(\tau_q)$ in a single invocation head $head(F(\tau_q))$, and a tail $tail(F(\tau_q))$, which contains the rest of the invocations of $F(\tau_q)$. $head(F(\tau_q))$ is unchanged, but $tail(F(\tau_q))$ is transformed to a pure QEDF task, as if the separated tail had been released immediately after the head. Unchanged head and changed tail constitute the newly formed $E(F(\tau_q))$.

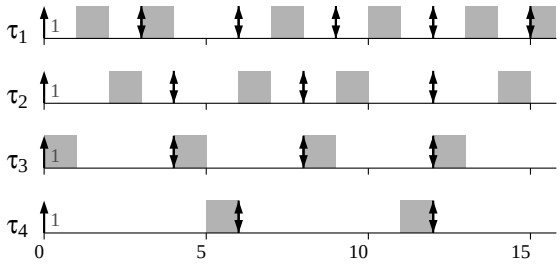
As an example, consider figure 3(a) where τ_3 is a normal QEDF task, in figure 3(b) it has changed to $F(\tau_3)$, and in figure 3(c) it has further changed to the EQT $E(F(\tau_3))$. Note that the tail of $E(F(\tau_3))$ is scheduled according to EDF rules.

In case of deadline ties a deterministic choice determines the order in which the tasks are executed. A simple criterion such *lowest task index first* will fit.

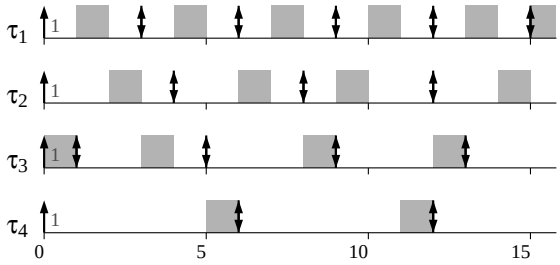
Theorem 4 (From QEDF task to PIT)
Given a feasible QEDF set Γ with utilisation $U \leq 1$. Any QEDF $\tau_q \in \Gamma$ can be pro-



(a) τ_3 is a normal quantum task



(b) τ_3 is a Push-In Task (PIT)



(c) τ_3 is an Early Quantum Task (EQT)

Figure 3: Synchronous quantum tasks of Γ_1 scheduled by EDF

moted to PIT $F(\tau_q)$ without deadline overflow of any other task in Γ.

Proof First we prove that the executions of τ_q may be shifted to the left until the first slot after its release under the condition $U < 1$ and secondly we prove the exceptional case for $U = 1$. Both parts are proved by construction, which is illustrated in the figures 3(a) and 3(b).

First part Denote release time, start time and deadline for the j^{th} quantum invocation of τ_q by r_q^j, s_q^j, and d_q^j respectively.

Call the quanta within invocation j from r_q^j until s_q^j *left-hand* slots and those after s_q^j until d_q^j the *right-hand* slots.

We will now rotate all left-hand slots to the right such that all executions except the last one are shifted one place to the right. The last one, containing s_q^j, is moved to the free falling first slot. None of the executions in the rotated list has passed its deadline or release time: for the executions shifted to the right there is a slack time of at least 1 due to theorem 3, and the execution of the PIT τ_q has been inserted in the first slot, immediately after its release time.

Furthermore, the right-hand sequence of executions is not affected by this operation. Consequently none of the executions in the quanta from r_q^j to d_q^j did violate release times or deadline requirements, while the load of τ_q is executed as early as possible. We can repeat construction and arguments for all executions of τ_q until we have constructed $F(\tau_q)$ which completes the proof of the first part.

Second part Here we consider the case where $U = 1$. Denote the start of the last execution before the hyper-period s_i^{last}, which is the only execution with a slack 0 in a hyper-period. There are two cases now: τ_i is promoted to periodic PIT, or a different task is promoted.

If τ_i itself would be promoted, then the last execution before the hyper-period of τ_i, is part of the left-hand side. This side is rotated and after the rotation this execution is mapped on r_i^{last}, while the other quanta of the left-hand side can be shifted to the right, without violating any of the deadlines of the other tasks, since they all have a slack of at least 1.

If τ_i itself is *not* subject of promotion to PIT, then the last execution τ_i^{last} before the hyper-period, will always be part of the right-hand side of invocations. Consequently this quantum is left unaffected by any rotation operation. This completes the construction of $F(\tau_i)$ and completes the proof of the second part and with it the proof of the theorem.

□

Example 3 (Task set Γ_1)

We have chosen to promote task τ_3 of the earlier presented task set Γ_1 in figure 2 to a PIT. According to figure 3(b) the following shifts of executions takes place:

$$\begin{array}{llll} 0 \rightarrow 1, & 1 \rightarrow 2, & 2 \leftarrow 0; \\ 4 \rightarrow 5, & 5 \rightarrow 6, & 6 \leftarrow 4; \\ 8 \rightarrow 9, & 9 \rightarrow 10, & 10 \leftarrow 8; \\ \ldots \end{array}$$

■

A PIT can be used as an aperiodic quantum server, quite in the spirit of the dynamic servers as presented in [6], with arrival intervals of aperiodic events that are not smaller than its period. It is in fact an improved total bandwidth server [14, 15] that can serve aperiodic events. It is even better than the improved total bandwidth server since no

computation is needed for the earliest release possible.

Theorem 5 (From PIT to EQT)
Given a set of quantum tasks Γ *with utilisation* $U \leq 1$. *One of its tasks can be promoted to EQT without deadline overflow of any other task in* Γ.

Proof Let τ_q be the task that will be promoted to EQT. We will proceed according to the following steps.

(1) According to theorem 4 τ_q can be transformed to PIT $F(\tau_q)$ without endangering any deadline in Γ.

(2) Next choose a head and tail such that the number of invocations in the head is 1. Successively adapt the tail to a normal QEDF task while maintaining feasibility with the following steps:

(2.a) Tail invocations $tail(F(\tau_q))$ are part of the PIT and consequently they can execute in the first slot immediately after their release. Therefore their deadlines can be set to 1.

(2.b) Release times of tail invocations can be set $T_q - 1$ earlier, that is just after the deadline of the previous invocation, without forcing any of the execution of the other tasks to a different time slot.

(2.c) Step (2.a) and (2.b) do not influence the feasibility. In this last step we make use of aperiodic EDF optimality arguments of Dertouzos [7]: "if there exists a feasible schedule for a task set then EDF is able to find it". As a consequence we can adapt the tail to a QEDF task while preserving feasibility. This completes the construction of $E(F(\tau_q))$ and completes the proof.
□

The construction from PIT to EQT is illustrated in figure 3(b) and 3(c), where τ_q is embodied by τ_3.

In the following section we introduce the maximum bandwidth PIT. This task is not really in the system but it is introduced in order to estimate the time needed to regain the slack or bandwidth that has been used by an early invocation.

5 EQT admission control

In this section we will illustrate admission control of EQTs. During the introduction of an EQT slack space is consumed for the early execution of the head. We will consider the executions of the last two heads in more detail and take a view as if these heads were executed by a virtual PIT that uses all remaining bandwidth (or utilisation) exclusively for this purpose. From the minimum period of the virtual PIT we can derive the minimum inter-arrival interval of the heads of the EQTs.

Let t_p be the time at which the previous EQT head of τ_p has been executed and let t_r be the moment at which a new EQT of τ_r with period T_r is requested. The utilisation after introduction of τ_p is U. The minimum period T_{vir} of a virtual PIT, at which cost the last head at t_p as well as the new head t_r can be executed, is determined by the utilisation $T_{vir}(U) = \lceil 1/(1-U) \rceil$. $T_{vir}(U)$ is therefore a safe estimation for the shortest inter-arrival time between two heads of successive arrivals. For instance if $U = 3/4$ after introduction of τ_p hence $T_{vir}(U) = 4$. Therefore the inter-arrival distance between t_p and t_r is 4.

Another meaningful or even better interpretation of $T_{vir}(U)$ is to consider it as the replenishment time for the consumed slack space of the previous head. The following corollary emphasises this interpretation explicitly:

Corollary 2 (Replenishment)
Given a feasible EQT set Γ *with utilisation* $U < 1$. *The replenishment interval or inter-arrival interval is equal to* $T_{vir}(U) = \lceil 1/(1-U) \rceil$.

From this the following admission control conditions can be concluded.

Corollary 3 (Admission conditions)
Given an EQT set Γ *with utilisation* $U \leq 1$. *Let* τ_r *be a new EQT.* τ_r *is admitted if*

$$T_r \geq \left\lceil \frac{1}{1-U} \right\rceil \tag{9}$$

and if the invocation of the previous EQT was at least

$$T_{vir}(U) = \left\lceil \frac{1}{1-U} \right\rceil \textit{earlier.} \tag{10}$$

If there is sufficient slack space, then equation 10 is too strict and more tolerant conditions can be established. These conditions are still under development.

In the following section we will show a typical application for an EQT server. It serves multimedia streams, which are started as soon as possible.

6 EQT server and buffer management

Consider a multimedia server which serves a set Γ of n streams $\{\tau_1 \dots \tau_n\}$. Depending on the bit rate of the consumer and on the properties of the server we need a double or a triple buffer for each stream in order to guarantee the absence of buffer over- or underflow. Both buffer models are valid in the context of this paper. Since the double buffer model uses less resources we use this model.

Transferring data from the server to a buffer is done in synchronous non-preemptive quantum time slices and scheduled by QEDF. For normal QEDF streams, the begin of filling the buffer is determined by the order of deadlines, which, in the worst case can be just before the deadline of the first invocation. This can be prohibitive with multimedia servers for streams with large periods, hence long deadlines.

Using EQTs is a solution to this problem. The server properties conform to the definition of a synchronous task, or streams, as given in section 3, so the n different client streams in Γ can be scheduled with an EQT server. Client i reads a buffer with a period T_i^c, a non-integer value, which is determined by the application, while the server writes the buffer with a period T_i:

$$T_i = \lfloor T_i^c \rfloor \geq 1 \tag{11}$$

The shorter period of the server implies that we need to take measures to prevent buffer overflow. We assume a double buffer system with a total buffer capacity of $2B$ with n streams $\tau_1 \dots \tau_n$. After having filled the first buffer with the head of the EQT stream τ_i, the consumption of the buffer can be started immediately, simultaneously with the start of the tail at the server side.

We define the buffer usage β_i as the amount of buffer space that is in use by stream τ_i, varying such that $\beta_i \in [0, 2B]$. Denote the j^{th} invocation of τ_i as τ_i^j, and the buffer usage at release time of invocation τ_i^j as β_i^j. At the start of the first tail invocation τ_i^1, the buffer usage β_i^1 is exactly B, filled by invocation τ_i^0, the head. Invocation τ_i^j is only started under the condition that there is enough buffer space available:

$$\beta_i^j \leq B \rightarrow \text{start}(\tau_i^j) \tag{12}$$

If, at *release* time of invocation τ_i^j, there is not enough buffer space ($\beta_i^j > B$), then the

release time r_i^j as well as the deadline d_i^j are delayed by 1 time unit.

Theorem 6 (No over- or underflow)
Given a double buffered PIT client/server with buffer capacity $2B$ and with a client bit rate consumption of period $T_i^c \geq 1$, and a server production of period $T_i = \lfloor T_i^c \rfloor$. The release and deadline of an invocation are delayed by one time unit if at release time the free buffer capacity is below B.

In such a system buffer over- or underflow cannot occur for any stream τ_i.

The proof of theorem 6 is given in appendix A. This proof also gives rise to the following corollary.

Corollary 4 (Single delay only)
In the system as described in theorem 6 no two successive conditional delays will occur.

Periodic jukebox

We have built a jukebox server. It is the core of a Video-on-Demand (VoD) system, which serves real-time audio and video streams to users distributed in a network. We investigated two types of servers for the jukebox, a periodic server and an aperiodic server. Both servers schedule media streams of users such that a stream can be started as soon as possible while maintaining the timely continuation of all streams admitted earlier. The aperiodic server does this by computing an aperiodic schedule and it computes a nearly optimal solution by solving a rather complex (NP-hard) scheduling problem. The periodic server may use EQT or QEDF scheduling. We found that a periodic service is not a good idea for a jukebox with a large disk cache. However, when such a cache cannot be afforded and one can only use limited buffer space, the EQTs perform better than normal QEDF tasks. For a description of the aperiodic server we refer to [10] and for the periodic server to [11].

7 Conclusion

In order to speed up the start of a new *Quantum EDF* task, we shrink its first period into one quantum slot. Such a task is called an *Early Quantum Task* (EQT) and its role is to be introduced as early as possible without exceeding any of the deadlines of the already running tasks. We have shown that a new EQT can be admitted if the utilisation $U < 1$, as if it were a normal EDF task, and that it may be started immediately if the system has sufficient *quantum slack* left. When a new EQT is introduced, one slack quantum is consumed for the early execution. The system regains this quantum automatically in $\lceil 1/(1-U) \rceil$ time units. The initial quantum slack estimation depends on the utilisation and on the shortest period T_{min} of the tasks in use and is equal to $\lceil (1-U)T_{min} \rceil$.

Next we have illustrated the use of the EQTs for a buffered multi-media server. We have shown that EQTs provide a considerable speedup of the start of the stream at the client side. Additionally we proved the correctness of a simple mechanism to avoid buffer over- and underflow.

Acknowledgements

Acknowledgements are due to Pieter Hartel for valuable comments.

References

[1] S. V. Anastasiadis, K. C. Sevcik, and M. Stumm. Server-based smoothing of variable bit-rate streams. In *Proceedings of the ninth ACM Multimedia Confer-*

ence, pages 147–158, Ottawa, October 2001.

[2] J. H. Anderson and A. Shrinivason. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pages 76–85, Delft, June 2001.

[3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. In *ACM Symposium on Theory of Computing*, pages 345–354, 1993.

[4] S. K. Baruah, A. K. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the Real-Time Systems Symposium*, pages 182–190, December 1990.

[5] Peter Bosch. *Mixed-media file systems.* PhD thesis, University of Twente, June 1999.

[6] Giorgio C. Buttazzo. *Hard Real-Time Computing systems - Predictable Scheduling Algorithms and Applications.* Kluwer Academic Publishers, 1997.

[7] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *Information Processing 74, proceedings of IFIP congress 74*, pages 807–813, Stockholm, Sweden, August 1974. North Holland Publishing Company. ISBN 0-7204-2803-3.

[8] K. Jeffay and D.L. Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 212–221, December 1993.

[9] J. H. M. Korst, V. Pronk, and P. Coumans. Disk scheduling for variable-rate data streams. In *Proceedings of the fourth international workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, IDMS '97*, volume LNCS 1309, pages 119–132. Springer-Verlag, September 1997.

[10] M. E. Lijding, P. G. Jansen, and S. J. Mullender. A flexible real-time hierarchical multimedia archive. In *Joint Int. Workshop on Interactive Distributed Multimedia Systems / Protocols for Multimedia Systems (IDMS/PROMS)*, page to appear, Coimbra, Portugal, Nov 2002. Springer-Verlag, Berlin.

[11] Maria Eva Lijding, Ferdy Hanssen, and Pierre Jansen. A case against periodic jukebox scheduling. Technical Report to appear, Centre for Telematics and Information Technology, University of Twente, 2002.

[12] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[13] M. Spuri. Analysis of deadline scheduled real-time systems. Research Report 2772, INRIA, January 1996.

[14] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the IEEE Real-time Systems Symposium*, 1994.

[15] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Journal of Real-time Systems*, 10(2), 1996.

A Proof of theorem 6

Proof (No buffer over- and underflow) During 1 time unit $\frac{1}{T_i^c}B$ is read from the buffer. This means that during invocation τ_i^j, which lasts T_i time units, $T_i\frac{1}{T_i^c}B$ is read from the buffer. Each invocation also fills the buffer once with B. Therefore, during each invocation the buffer usage β grows with $B - T_i\frac{1}{T_i^c}B$, for which holds:

$$0 \leq B - T_i\frac{1}{T_i^c}B < \frac{1}{T_i^c}B \qquad (13)$$

We derive equation 13 next, in three steps:

1. From equation 11 follows:

$$\frac{1}{T_i+1} < \frac{1}{T_i^c} \leq \frac{1}{T_i} \qquad (14)$$

2. First we derive $0 \leq B - T_i\frac{1}{T_i^c}B$ from equation 14:

$$\begin{aligned}
&\frac{1}{T_i^c} \leq \frac{1}{T_i} \implies \\
&T_i\frac{1}{T_i^c} \leq 1 \implies \\
&0 \leq 1 - T_i\frac{1}{T_i^c} \implies \\
&0 \leq B - T_i\frac{1}{T_i^c}B
\end{aligned}$$

3. Next we derive $B - T_i\frac{1}{T_i^c}B < \frac{1}{T_i^c}B$ from equation 14:

$$\begin{aligned}
&\frac{1}{T_i+1} < \frac{1}{T_i^c} \implies \\
&T_i^c < T_i + 1 \implies \\
&T_i^c - T_i < 1 \implies \\
&1 - T_i\frac{1}{T_i^c} < \frac{1}{T_i^c} \implies \\
&B - T_i\frac{1}{T_i^c}B < \frac{1}{T_i^c}B
\end{aligned}$$

Thus follows equation 13.

Buffer overflow Invocation τ_i^j is only started under the condition that there is enough buffer space available (equation 12). By using conditional delays this is guaranteed (see section 6). At the deadline of invocation τ_i^j the new buffer usage β_i^{j+1} will be $\beta_i^j + B - T_i\frac{1}{T_i^c}B$. Buffer overflow cannot occur, because:

$$\begin{aligned}
&\beta_i^{j+1} = \beta_i^j + B - T_i\frac{1}{T_i^c}B \overset{\text{by (12)}}{\implies} \\
&\beta_i^{j+1} \leq B + B - T_i\frac{1}{T_i^c}B \overset{\text{by (13)}}{\implies} \\
&\beta_i^{j+1} < B + \frac{1}{T_i^c}B \overset{\text{by 11}}{\implies} \\
&\beta_i^{j+1} < 2B
\end{aligned}$$

Buffer underflow Since invocation τ_i^j is started only when there is enough buffer space ($\beta_i^j \leq B$), we have to make sure no buffer underflow can occur. If no buffer underflow will occur when the invocation is executed at the latest possible moment in time, which is directly before the deadline, buffer underflow will also not occur when the invocation is executed earlier than that time. Say invocation τ_i^j is started at time $d_i^j - 1$, so it will just make its deadline (note: $C_i = 1$). At this moment in time, the consumer will have read $(T_i - 1)\frac{1}{T_i^c}B$. During the following time unit the buffer will start filling up, so buffer underflow will not occur if there is enough data in the buffer before the consumption at the beginning of r_i^j:

$$\beta_i^j \geq (T_i - 1)\frac{1}{T_i^c}B \qquad (15)$$

We will prove equation 15 by recursion:

1. At the start of consumption $\beta_i^1 = B$, which satisfies the condition, as derived

from equation 14:

$$\frac{1}{T_i^c} \leq \frac{1}{T_i} \implies$$
$$T_i \frac{1}{T_i^c} \leq 1 \overset{\text{by (11)}}{\implies}$$
$$(T_i - 1)\frac{1}{T_i^c} < 1 \implies$$
$$(T_i - 1)\frac{1}{T_i^c} B < B = \beta_i^1$$

2. When at release time r_i^j of invocation τ_i^j equation 15 holds, it will also hold at deadline time d_i^j. During this time the buffer usage β grows with $B - T_i \frac{1}{T_i^c} B$ to the new buffer usage β_i^{j+1}.

   (a) When $\beta_i^j \leq B$ at release time, the release is not delayed. Then:

   $$\beta_i^{j+1} = \beta_i^j + B - T_i \frac{1}{T_i^c} B \overset{\text{by (13)}}{\implies}$$
   $$\beta_i^{j+1} \geq \beta_i^j \overset{\text{by (15)}}{\implies}$$
   $$\beta_i^{j+1} \geq (T_i - 1)\frac{1}{T_i^c} B$$

   (b) When $\beta_i^j > B$ at release time, the release is delayed, and β_i^j shrinks with $\frac{1}{T_i^c} B$. So, $\beta_i^j > B - \frac{1}{T_i^c} B$ when invocation τ_i^j is actually released. This implies that equation 15 still holds at release time, as derived from equation 14:

   $$\frac{1}{T_i^c} \leq \frac{1}{T_i} \implies$$
   $$T_i \leq T_i^c \implies$$
   $$T_i - 1 \leq T_i^c - 1 \implies$$
   $$(T_i - 1)\frac{1}{T_i^c} \leq 1 - \frac{1}{T_i^c} \implies$$
   $$(T_i - 1)\frac{1}{T_i^c} B \leq B - \frac{1}{T_i^c} B < \beta_i^j$$

3. From item 1 and 2 equation 15 can be concluded.

This completes the proof.
□