

Jitter-Approximation Tradeoff for Periodic Scheduling

Zvika Brakerski
Dept. of Electrical Engineering
Tel Aviv University
Tel Aviv 69978
Israel
zvika@eng.tau.ac.il

Boaz Patt-Shamir *
HP Cambridge Research Lab.
One Cambridge Center
Cambridge, MA 02142
USA
boaz@hp.com

Abstract

We consider an asymmetric wireless communication setting, where a server periodically broadcasts data items to different mobile clients. The goal is to serve items in a prescribed rate, while minimizing the energy consumption of the mobile users. Abstractly, we are presented with a set of jobs, each with a known execution time and a requested period, and the task is to design a schedule for these jobs over a single shared resource without preemption. Given any solution schedule, its period approximation is the maximal factor by which the average period of a job in the schedule is blown up w.r.t. its requested period, and the jitter ratio is roughly the maximal variability of times between two consecutive occurrences of the same job. Schedules with low jitter ratio allow the mobile devices to save power by having their receivers switched off longer. In this paper we consider a scenario where clients may be willing to settle for non-optimal period approximation so that the jitter ratio is improved. We present a parametric jitter-approximation tradeoff algorithm that allows us to choose various combinations between jitter optimality and period optimality for any given set of jobs.

1. Introduction

In broadcast disks [1], a powerful server broadcasts data items to mobile clients awaiting their desired items (e.g., public data like stock quotes, or user data like an address book). In an arbitrary broadcasting schedule, a client may have to “busy-wait” for its item, i.e., actively listen to the server until its item is broadcast, thus wasting much battery power. If the broadcast schedule is *perfectly periodic*, i.e., each item i is broadcast precisely every p_i time units for some p_i , then the client can switch on its radio exactly

when needed. However, an egalitarian round-robin schedule (which is perfectly periodic) is not satisfactory: a general solution must also accommodate for a different periodicity requirement for each item, since different items may have different popularity levels with clients, different expiration times, different QoS levels etc.

Broadcast disks are just one example among many where it is desirable to have low *jitter*, namely the spacing between consecutive occurrences of the same item should be as equal as possible. Another example from the wireless world is the Sniff Mode in Bluetooth [3]. In this case, slave devices can shut off their transceivers except for a certain time every once in a while, when they listen to find out whether the master device is trying to contact them. If the master uses a schedule with low jitter, it would help improving battery lifetime in the slave devices.

In this paper, we present an algorithmic study of such scenarios. To allow us to ignore idiosyncrasies of any particular technology, we consider the following abstract model (formal definition is provided in Section 2). An *instance* of the problem consists of a set of *jobs*, where each job has known *length* and *requested period*. The task is to design a single-server non-preemptive periodic schedule of the jobs, i.e., each job is assigned an infinite sequence of occurrence times such that no two distinct occurrences of any two jobs overlap. The *granted period* of a job in a schedule is the average time between two consecutive occurrences of that job. Jitter is defined as follows. Consider the set of all interval lengths between two consecutive occurrences of a job. (The average of these lengths is the granted period of the job.) The *jitter ratio* of that job is the maximal difference between such an interval length and the granted period, normalized by dividing the difference by the granted period. In this work, we evaluate a schedule by its worst case period approximation, and its worst-case jitter ratio over all jobs. We would like the schedule to have the smallest possible period approximation (1 means that each granted period is no larger than the corresponding requested period), and the

*On leave from Dept. of Electrical Engineering, Tel Aviv University.

smallest possible jitter ratio (0 is a trivial lower bound that holds iff the schedule is perfectly periodic).

Constructing schedules with period approximation 1 is a well-studied problem, starting with the seminal work of Liu and Layland [6]. Unfortunately, there are cases where insisting on period approximation 1 implies that the jitter ratio can be as high as 1, i.e., the job can occur at anytime, which means in the wireless context that the receivers might need to stay powered all the time. On the other extreme, there are a few algorithms that construct perfectly-periodic schedules (with jitter ratio 0), but they cannot have period approximation 1. As a quick example to that effect, consider an instance that contains two jobs (among others), each of unit length, such that one job requests period 2 and the other requests period 3. By the Chinese Remainder Theorem, any schedule with these periods will have these two jobs collide every 6 time slots, and hence it cannot be the case that the jitter ratio is 0 and the period approximation is 1 simultaneously.

In this paper, we try to win (most of) the good of both worlds by developing an algorithm that allows one to trade jitter ratio for period approximation. One way to use this algorithm is to feed it with an instance and a parameter that specifies the maximal allowed jitter ratio; the algorithm then outputs a periodic schedule for this instance that (1) satisfies the jitter ratio parameter, and (2) has period approximation guarantee better than the best previously known bounds.

What’s known. Motivated by operating systems and other centralized scheduling scenarios, most previous work about periodic scheduling took the viewpoint that period approximation must not be larger than 1, and jitter is only of secondary importance. For example, Liu and Layland [6] define periodic scheduling to be one where a job with period τ is scheduled exactly once in each time interval of the form $[(k-1)\tau, k\tau-1]$ for any integer k . Naïvely interpreted, this definition allows jitter ratio as high as 1, which is not useful. Baruah *et al.* [2] still insist on keeping the period approximation 1, but try to minimize jitter. They define a generalized concept of jitter, prove bounds on the jitter ratio in terms of the specific instance at hand, and propose algorithms that search for schedules with minimal jitter under this restriction. In this paper, we use a special case of their definition (they allow arbitrarily weighted jitter). Cast into our language, their jitter bound is as follows. Let β_i denote the *bandwidth* request of job i , defined to be the job length of i divided by its period, and let $\beta = \sum_i \beta_i$ over all jobs i in the instance. Then the jitter ratio of a job i is at most $1 - (\beta - \beta_i)$.

General perfectly periodic schedules are defined and analyzed in [4], where the concept of the *extent* of an instance is defined. Formally, the extent of an instance, denoted R , is the ratio between the maximal job length and the shortest

job period. It is proved that any perfectly periodic schedule has period approximation at least $1 + R$, and an algorithm with approximation ratio $1 + O(R^{1/3})$ is presented (note that $R < 1$, so $R^{1/3} > R$). Naaman and Rom [7] study the case where the ratio between periods of jobs is always an integer. They give an algorithm to generate schedules with period approximation 1 and jitter ratio $(k-1)R$, where k is the number of distinct requested periods and R is the extent of the instance. They show that this bound is tight for period approximation 1.

Our results and paper organization. In this paper we present an algorithm that, given any instance of periodic scheduling, and an integer parameter g , produces a schedule with period approximation less than $1 + \frac{\sqrt{2}}{2} + R/2^{g-1} \approx 1.707 + R/2^{g-1}$ and jitter ratio at most Rg . The parameter g must be non-negative and cannot be larger than $\log_2 \frac{T}{t}$, where T and t are the largest and smallest requested periods. Incidentally, this algorithm, when applied with $g = 0$, improves on the best known results for perfectly periodic schedules for $R > 0.006$ [4]. Our algorithm is presented in two steps. First, in Section 3, we present Algorithm `cont_bal` that guarantees approximation ratio of $1 + R/2^g$ and jitter ratio of Rg , but this algorithm applies only to instances in which the ratio of any two periods is a power of 2. Using Algorithm `cont_bal` as a subroutine, we specify in Section 4 our final algorithm, which applies to any instance.

The formal model is presented next, in Section 2. Some concluding remarks are given in Section 5.

2. Problem Statement and Notation

Most of the notation used in this work is summarized in the Glossary in Figure 1.

Instances. An *instance* of the perfectly-periodic scheduling problem is a set of n jobs $\mathcal{J} = \{j_i\}_{i=1}^n$, where each job $j_i = (b_i : \tau_i)$ has *length* (or *execution time*) b_i , and *requested period* τ_i . We sometimes refer to jobs also as *clients*. The maximal length of a job in an instance \mathcal{J} is denoted by $B_{\mathcal{J}} \stackrel{\text{def}}{=} \max\{b_i \mid i \in \mathcal{J}\}$. The maximal and minimal values of the requested periods in instance \mathcal{J} are denoted by $T_{\mathcal{J}} \stackrel{\text{def}}{=} \max\{\tau_i \mid j_i \in \mathcal{J}\}$, and $t_{\mathcal{J}} \stackrel{\text{def}}{=} \min\{\tau_i \mid j_i \in \mathcal{J}\}$. The ratio between $B_{\mathcal{J}}$ and $t_{\mathcal{J}}$ is called the *extent* of \mathcal{J} , formally defined by $R_{\mathcal{J}} \stackrel{\text{def}}{=} \frac{B_{\mathcal{J}}}{t_{\mathcal{J}}}$. The *requested bandwidth* of job j_i is defined by $\beta_i \stackrel{\text{def}}{=} \frac{b_i}{\tau_i}$. The total bandwidth of an instance \mathcal{J} is defined by $\beta_{\mathcal{J}} \stackrel{\text{def}}{=} \sum_{i=1}^n \beta_i$. We assume that $\beta_{\mathcal{J}} \leq 1$ always. The *free bandwidth* of an instance \mathcal{J} is defined by $\Delta_{\mathcal{J}} \stackrel{\text{def}}{=} 1 - \beta_{\mathcal{J}}$. We omit the subscript \mathcal{J} when the instance is clear from the context.

Instances and jobs:

- \mathcal{J} : an instance of the problem.
- n : number of jobs (clients) in an instance.
- j_i : the i th job in an instance.
- b_i : length (execution time) of j_i .
- τ_i : requested period of j_i .
- $B_{\mathcal{J}} \stackrel{\text{def}}{=} \max \{b_i \mid i \in \mathcal{J}\}$
- $T_{\mathcal{J}} \stackrel{\text{def}}{=} \max \{\tau_i \mid i \in \mathcal{J}\}$
- $t_{\mathcal{J}} \stackrel{\text{def}}{=} \min \{\tau_i \mid i \in \mathcal{J}\}$
- $R_{\mathcal{J}} \stackrel{\text{def}}{=} \frac{B_{\mathcal{J}}}{t_{\mathcal{J}}}$: extent of instance \mathcal{J} .

- $\beta_i \stackrel{\text{def}}{=} \frac{b_i}{\tau_i}$: requested bandwidth of j_i .
- $\beta_{\mathcal{J}} \stackrel{\text{def}}{=} \sum_{i \in \mathcal{J}} \beta_i$: total bandwidth of instance \mathcal{J} .

Schedules and quality measures:

- S : a schedule.
- τ_i^S : granted period of j_i in schedule S .
- $\rho_i \stackrel{\text{def}}{=} \frac{\tau_i^S}{\tau_i}$: period approximation of j_i in schedule S .
- $\rho(\mathcal{J}, S) \stackrel{\text{def}}{=} \max \{\rho_i \mid j_i \in \mathcal{J}\}$: period approximation of S w.r.t. \mathcal{J} .
- σ_i : Jitter ratio of job i .
- $\sigma(S) \stackrel{\text{def}}{=} \max \{\sigma_i \mid j_i \in \mathcal{J}\}$: Jitter ratio of S .

Figure 1: Glossary of notation.

Schedules. A *schedule* S for an instance \mathcal{J} is an infinite sequence of *start times* s_0, s_1, s_2, \dots , where each start time s_k is mapped to a job $j(k) \in \mathcal{J}$. We say that job $j(k)$ is *scheduled* at the time slots $s_k, s_k + 1, \dots, s_k + b_{j(k)} - 1$. A schedule is *feasible* only if no two jobs are ever scheduled at the same time step, i.e., for all $k \geq 0$, $s_{k+1} \geq s_k + b_{j(k)}$. A schedule is *cyclic* if it is an infinite concatenation of a finite schedule C , called the *cycle* of S . In this paper we consider only cyclic schedules.

Fix a feasible schedule S for an instance \mathcal{J} , and let C be the cycle of S . Assume without loss of generality that each job of \mathcal{J} is scheduled at least once in C . The *granted period* of a job j_i in S , denoted τ_i^S , is the number time slots in C divided by the number of start times of j_i in C . Note that the granted periods may be different from the requested periods, but the job lengths cannot be truncated by the schedule.

Given an instance \mathcal{J} with schedule S , the *period approximation* of a job j_i in S is $\rho_i \stackrel{\text{def}}{=} \frac{\tau_i^S}{\tau_i}$. The period approximation of S with respect to \mathcal{J} is $\rho(\mathcal{J}, S) \stackrel{\text{def}}{=} \max \{\rho_i \mid i \in \mathcal{J}\}$. To define jitter, let s_1, s_2, \dots be the start times of a job j_i in S , and let τ_i^S be its average period. The *jitter ratio* of j_i in S is $\sigma_i \stackrel{\text{def}}{=} \max_k \left\{ \frac{|(s_{k+1} - s_k) - \tau_i^S|}{\tau_i^S} \right\}$, and the jitter ratio of S is $\sigma(S) \stackrel{\text{def}}{=} \max_i \{\sigma_i \mid j_i \in \mathcal{J}\}$.

All logarithms in this paper are to base 2.

3. The Controlled Balance Algorithm

In this section we present our basic algorithm for periodic scheduling with controllable jitter, which works only when the ratio between any two periods is a power of two. The idea in the algorithm is to spread the jobs evenly over the schedule in a recursive fashion. The algorithm also adds idle time slots, at a level specified by the user, so as to reduce the jitter caused by possibly imperfect balancing. The algorithm is based on a known algorithm used for perfectly periodic schedules [4], augmented here with a way to control jitter by adding idle time slots. It has recently been brought to our attention that an algorithm similar to the one of [4] appears in [5] for general periodic scheduling; it is described as a heuristic without analysis, and without the controlled jitter idea.

The algorithm is given an integer parameter g such that $0 \leq g \leq \log \frac{T}{t}$ (recall that T is the longest requested period, t is the shortest requested period, and that their ratio is a power of 2). This parameter controls the tradeoff between jitter and approximation: setting $g = 0$ will result with a perfectly periodic schedule and poor approximation guarantee, and increasing g results in better approximation and poorer smoothness. For convenience, we also use the complementary parameter $h \stackrel{\text{def}}{=} \log \frac{T}{t} - g$.

Pseudo code for the algorithm is presented in Figure 2, and an example execution is depicted in Figure 4. Informally, the algorithm constructs a cycle of the schedule by allocating start times in a balanced way. This is done using a binary tree of $1 + \log \frac{T}{t}$ levels, whose leaves represent sub-intervals of the schedule cycle. Each node in the tree

Algorithm cont_bal

Input: Instance \mathcal{J} , parameter $0 \leq g \leq \log \frac{T}{t}$. Define $h \stackrel{\text{def}}{=} \log \frac{T}{t} - g$.

Output: A cycle of a schedule S for \mathcal{J} .

Code:

- (1) Construct a complete binary tree of $1 + \log \frac{T}{t}$ levels $0, 1, \dots, \log \frac{T}{t}$. Create a job replica for each job in the instance, and associate these replicas with the root.
 - (2) Traverse levels $0, 1, \dots, \log \frac{T}{t} - h - 1 = h - 1$ of the tree, in breadth-first order, starting from the root. In each visited node v , do $\text{split}(v)$.
 - (3) Let w be the maximal bandwidth associated with a node at level h . Add to each node at level h “dummy” job replicas to make all nodes have bandwidth w . Each dummy job replica has length 1 and period T .
 - (4) Traverse levels $h, \dots, \log \frac{T}{t} - 1$ of the tree, in breadth-first order. In each visited node v , do $\text{split}(v)$.
 - (5) Scan the leaves left-to-right. For each leaf ℓ , output the job replicas associated with ℓ in increasing \prec order, where dummy job replicas are output at the end of each leaf; they correspond to idle time slots.
-

Figure 2: Algorithm cont_bal.

contains “job replicas” derived from the original instance, where each job replica has its own associated period. The tree is constructed in a top-down fashion as follows. Initially, the root contains all job replicas that are exactly the jobs in the instance (Step 1). Each node has two children, whose replicas are defined by subroutine split (Steps 2 and 4). Pseudo-code for subroutine appears in Fig. 3. To ensure low jitter, split uses a total order on jobs, “ \prec ”, defined below. This order ensures that at any given level, for any given job j , all nodes in which a job replica associated with j occurs have the *same* set of job replicas preceding it. Put differently, a job has the same offset within a node for all its replicas at a level. In addition, nodes at level h are padded with “dummy” job replicas so that all nodes at level h have *exactly* the same length (Step 3). Finally, the leaves are scanned and their associated replicas are output (Step 5). The dummy replicas correspond to idle time slots. The consequence of the padding at level h is that the leaf lengths are *roughly* the same.

The \prec order is defined as follows. For jobs j_i, j_k with requested periods τ_i and τ_k , respectively, we say that $j_i \prec j_k$ if either $\tau_i < \tau_k$, or if $\tau_i = \tau_k$ and $i < k$. We stress that the “ \prec ” relation is defined on jobs, not job replicas. To extend the order to replicas, each job-replica uses the rank it inherits from its original job (even though a replica has a possibly different period).

Subroutine split

Input: A node v with its set of associated job replicas.

Output: Two sets of job replicas, associated with the children of v .

Code:

- (1) Scan the job replicas associated with v in increasing “ \prec ” order. Let j be the currently scanned replica, with period τ_j and size b_j :
 - (1a) If $\tau_j < T$, add identical job replicas to both children of v , each with size b_j and period $2 \cdot \tau_j$.
 - (1b) If $\tau_j = T$, add a job replica with period τ_j and size b_j to the child of v whose current total associated bandwidth (i.e., sum of the job replica lengths divided by their periods) is smaller. In case of a tie, add the replica to the left child.
-

Figure 3: Subroutine split.

The main properties of Algorithm cont_bal are summarized in the following theorem.

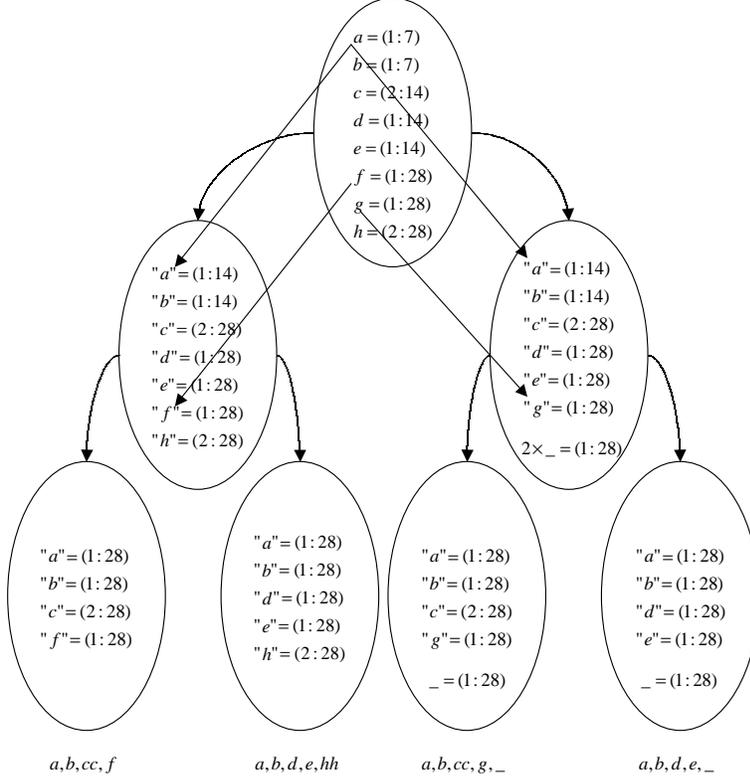
Theorem 3.1 *Let $\mathcal{J} = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance with extent R . Suppose that there exists a constant $c > 0$ such that for all i , $\tau_i = c \cdot 2^{e_i}$, where e_1, \dots, e_n are non-negative integers. Then Algorithm cont_bal with parameter g outputs a schedule S with $\rho(\mathcal{J}, S) \leq 1 + R/2^g$ and $\sigma(S) \leq Rg$.*

The theorem follows directly from Lemma 3.3 and Lemma 3.6 proved below. But first, note that the tradeoff between jitter and approximation is controlled by the value of g : a small value of g means better jitter and worse approximation. The extreme points are $g = 0$ (jitter ratio 0 and period approximation $1 + R$), and $g = \log \frac{T}{t}$ (period approximation 1 and jitter ratio $R \log \frac{T}{t}$).

We start the analysis with the following important property.

Lemma 3.2 *The bandwidth associated by Algorithm cont_bal with each node at level i is at most $2^{-i}(1 - \Delta) + (1 - 2^{-i})B/T$.*

Proof: We prove the claim by induction on i . For $i = 0$ the claim is trivial: the bandwidth associated with the root is $1 - \Delta$ by Step 1 of Algorithm cont_bal. Consider a node at level $i + 1$, and let β_1 denote its bandwidth. Let β_0 denote the bandwidth of its parent at level i , and let β_2 denote the bandwidth of its sibling. By induction hypothesis, $\beta_0 \leq 2^{-i}(1 - \Delta) + (1 - 2^{-i})B/T$. By the code of Algorithm split, the difference between the bandwidth associated with any two siblings is at most the bandwidth of one job replica whose period is T . Since the size of any job replica is at



Cycle : $a, b, cc, f, a, b, d, e, hh, a, b, cc, g, -, a, b, d, e, -$

Figure 4: An example run of Algorithm `cont_bal` with $g = 1$. The jobs of the original instance can be seen in the root node; “-” indicates a dummy replica. Some splits are indicated by straight arrows.

most B , we get $|\beta_1 - \beta_2| \leq B/T$. Assume without loss of generality that $\beta_1 \geq \beta_2$. Then we have that

$$\begin{aligned} \beta_1 &\leq \frac{\beta_0 + B/T}{2} \\ &\leq \frac{1}{2} (2^{-i}(1 - \Delta) + (1 - 2^{-i})B/T + B/T) \\ &= 2^{-i-1}(1 - \Delta) + (1 - 2^{-i-1})B/T, \end{aligned}$$

proving the inductive step for Step 2. Step 3 does not increase the maximal bandwidth associated with nodes at level h , the induction hypothesis is not violated for $i = h$, and hence the lemma holds also for Step 4. ■

Using Lemma 3.2, we bound below the period approximation. The result is in fact sharper than the bound stated in Theorem 3.1.

Lemma 3.3 *The period approximation of a schedule produced by `cont_bal` with parameter g is at most*

$$1 - \Delta + R/2^g - B/T.$$

Proof: Consider a job j_i with requested period $\tau_i = T/2^{e_i}$ for some integer $e_i \geq 0$. The average granted period of j_i

is its number of occurrences in a cycle of the schedule, divided by the length of the cycle. The number of start times of j_i in the output cycle is clearly 2^{e_i} , because in the final schedule, the number of start times is the number of job replicas corresponding to j_i in the leaves, and because all job replicas in the leaves have period T . It remains to bound the number of time slots in the output cycle. This is precisely the sum of the lengths of job replicas in the leaves, which, in turn, is T times the total bandwidth associated with the leaves. Now, the total bandwidth of leaves is the same as the total bandwidth of all nodes at level h , because no new bandwidth is added after Step 3. By Lemma 3.2, the bandwidth associated with each node at level h after Step 3 is at most $2^{-h}(1 - \Delta) + (1 - 2^{-h})B/T$. Multiplying by 2^h (the number of nodes at level h) we get a bound on the total bandwidth at the leaves, and further multiplying by T we get a bound on the number of time slots in the output schedule. We thus get that the output cycle contains at most $T(1 - \Delta) + (2^h - 1)B$ time slots. It therefore follows that the average period of j_i in the schedule is $2^{-e_i}(T(1 - \Delta) + (2^h - 1)B)$. Recalling that the requested period is $T/2^{e_i}$, and since j_i is an arbitrary job, we

have that the approximation factor of the schedule is at most

$$\begin{aligned}
\frac{T(1-\Delta)+(2^h-1)B}{T2^{-e_i}} &= 1 - \Delta + (2^h - 1)B/T \\
&= 1 - \Delta + \left(\frac{T}{t2^g} - 1\right) \frac{B}{T} \\
&= 1 - \Delta + 2^{-g}R - \frac{B}{T}. \quad \blacksquare
\end{aligned}$$

Remark: Lemma 3.3 shows that it may be the case that the period approximation is actually smaller than 1 (i.e., all jobs get larger share than they asked for). This happens when $\Delta > R/2^g$, i.e., when there is a lot of free bandwidth in \mathcal{J} . If this is undesirable, we can scale up the all periods by a factor of $1 - \Delta + 2^{-g}R$ and apply the algorithm to the scaled instance. Note that the jitter bound would actually decrease in this case by the scaling factor, since t will grow by that factor and B remains fixed.

To analyze the jitter of the schedules produced by `cont_bal`, we need the following key observation.

Lemma 3.4 *Consider the set of job replicas associated with each node as a list sorted in increasing \prec order. Suppose that two job replicas j', j'' of the same job j are associated with two nodes v', v'' in the same level. Then the same job replicas precede j' in the ordered list of v' and precede j'' in v'' .*

Proof: Focus on a single split operation. There are two cases to consider. If a job replica j at the parent appears at both children as j' and j'' , then the period of j (at the parent) is smaller than T . In this case, by definition of \prec , all jobs preceding j at the parent also have periods smaller than T , and hence they will appear at both children, preceding both j' and j'' . In the second case, j has period T at the parent, and hence it will appear in only one child. In this case, the identity of the child and j 's rank in that child's sorted list depend only on the job replicas preceding j in the parent's list. Applying induction completes the proof. \blacksquare

We also need the following lemma.

Lemma 3.5 *Let W_h be the bandwidth associated with nodes at level h after Step 3. Then for any $0 \leq i \leq g$, the bandwidth associated with any node at level $h+i$ is at least $2^{-i}W_h - (1-2^{-i})B/T$ and at most $2^{-i}W_h + (1-2^{-i})B/T$.*

Proof: The upper bound is just a restatement of Lemma 3.2. The proof of the lower bound is repeating the arguments of the proof of Lemma 3.2, with the basis of the induction replaced by the fact that all nodes at level h have the same bandwidth after Step 3. \blacksquare

Lemma 3.6 *The jitter ratio in the schedule produced by `cont_bal` with parameter g is at most Rg .*

Proof: Consider a job j_i with period $\tau_i = T/2^{e_i}$ for some integer $e_i \geq 0$. By the algorithm, there will be a job replica in each node of level e_i , with associated period T . By Lemma 3.4, the list of job replicas preceding the job replicas of j_i is the same in all these nodes. Consider now the subtrees rooted at the nodes at level e_i . Clearly, exactly one job replica will appear in the leaves of a each sub-tree. Number the leaves of the subtrees by $0, 1, \dots, 2^{e_i} - 1$ from left to right. Since the allocation of a job replica to a child depends, by Algorithm split, only on the jobs preceding it in the \prec order, we conclude that a job replica of j_i will be placed in leaf number k in any subtree if and only if it is placed in leaf number k in all subtrees. Moreover, by Lemma 3.4, the set of job replicas preceding j_i in each leaf will be the same. It follows that the variability in the time between consecutive occurrences of j can be caused by leaves of different sizes. So consider the start time of a leaf. We claim that the start time of leaf number k is at least $kT2^{-g}W_h - gB$ and at most $kT2^{-g}W_h + gB$ time units after the start of a subtree rooted at a node at level e_i , where W_h is the bandwidth of nodes at level h . To see why this is true, note that the start time of leaf k is exactly the sum of the bandwidths of leaves $0, \dots, k-1$ times T . Consider the path from the root to leaf k : this path contains nodes which are left and right children. The key observation is that the total bandwidth of leaves preceding k is exactly the sum of bandwidths of nodes which are the left siblings of right-children nodes in the path leading to k . By construction, the total bandwidth of a tree rooted at level $i \leq h$ is $2^{h-i}W_h$. By Lemma 3.5, the total bandwidth of a tree rooted at level $i > h$ is at least $2^{h-i}W_h - B/T$ and at most $2^{h-i}W_h + B/T$. Since there are at most g nodes which are right children on the path leading to k , we get that the maximal time between two consecutive occurrences of j_i is $\tau_i + Bg$, and the minimal time between two consecutive occurrences of j_i is $\tau_i - Bg$. Hence $\sigma_i \leq 1 + Bg/\tau \leq Rg$. \blacksquare

4. Algorithm for General Instances

Algorithm `cont_bal` requires that the ratio between any two periods to be a power of 2. In this section we lift this restriction, and consider general instances.

One straightforward way to do that (suggested, e.g., in [5]) is to round all requested periods up to the next power of 2. This immediately gives us, for instances whose requested bandwidth is at most 1, guaranteed period approximation of at most $2 + R/2^{g-1}$ and jitter ratio of at most Rg . However, more judicious rounding allows us to obtain substantially better period approximation, and even to break the barrier of 2 for instances with small extent. Specifically, in this section we present an algorithm that guarantees, for any instance, period approximation less than $1.71 + R/2^{g-1}$ and

Algorithm B

Input: Instance \mathcal{J} , parameter g .

Output: Cycle for schedule S .

Code:

- (1) Let $\tau'_i = 2^{\lceil \log \tau_i \rceil}$, $t' = \min \{\tau'_i\}$. Execute Algorithm cont_bal on $\{(b_i : \tau'_i)\}$ with parameter g and denote the result by S_1 .
 - (2) Let $\tau''_i = 2^{\lceil \log(\tau_i) - \frac{1}{2} \rceil}$, $t'' = \min \{\tau''_i\}$. Execute Algorithm cont_bal on $\{(b_i : \tau''_i)\}$ with parameter g and denote the result by S_2 .
 - (3) If $\rho(\mathcal{J}, S_1) \leq \rho(\mathcal{J}, S_2)$, return S_1 . Otherwise, return S_2 .
-

Figure 5: Algorithm B.

jitter ratio at most Rg . Moreover, this approximation is obtained using only powers of 2 times a common multiple as periods in the final schedule.

The algorithm, called Algorithm B, is presented in Figure 5. The idea is as follows. The algorithm tries two forms of rounding. The first (Step 1) is to round *up* each period τ_i to the next power of 2, and the second (Step 2) is to round each period τ_i to the *closest* power of 2: periods between $2^{k-1/2}$ and $2^{k+1/2}$ are rounded to 2^k . Algorithm cont_bal is applied to both rounded instances, and the schedule with the better period approximation of the two alternatives is the final output (Step 3). Below, we prove that at least one of the two schedules has period approximation not larger than $1 + \frac{\sqrt{2}}{2} + R/2^{g-1} \approx 1.707 + R/2^{g-1}$.

Theorem 4.1 *Let $\mathcal{J} = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of the scheduling problem with requested bandwidth at most 1, and let S be the schedule produced for \mathcal{J} by Algorithm B with parameter g . Then $\rho(\mathcal{J}, S) \leq 1 + \frac{\sqrt{2}}{2} + R/2^{g-1}$ and $\sigma(S) \leq Rg$.*

Proof: Consider Step 1. Denote the bandwidth of the rounded instance $\{j_i = (b_i : \tau'_i)\}_{i=1}^n$ by β' , its extent by R' , and its free bandwidth by $\Delta' = 1 - \beta'$. We proceed by case analysis, depending on the relation between Δ' and a certain threshold parameter δ_0 whose value will be specified later.

Intuitively, the idea is as follows. If $\Delta' \geq \delta_0$, we have “a lot” of free bandwidth, and hence S_1 will have a good period approximation. If $\Delta' < \delta_0$, i.e., there is “little” free bandwidth, then it must be the case that only a small fraction of the bandwidth is associated with jobs whose periods were rounded up by more than $\sqrt{2}$. Therefore, we can afford to round the periods of these jobs *down*, thus getting an instance where no period was rounded up by much, and S_2 will have a good period approximation.

Formally, we argue as follows.

Case 1: $\Delta' \geq \delta_0$. Consider the application of Algorithm cont_bal to the rounded instance $\{(b_i : \tau'_i)\}$. By Theorem 3.1, the algorithm gives an period approximation of at most $1 - \Delta' + 2^{-g}R' \leq 1 - \delta_0 + 2^{-g}R$. Since the rounding changes the periods in S_1 by a factor of less than 2, we can conclude that in this case,

$$\rho(\mathcal{J}, S_1) \leq 2 - 2\delta_0 + 2^{1-g}R. \quad (1)$$

The jitter ratio in this case is $R'g \leq Rg$.

Case 2: $\Delta' < \delta_0$. In this case we concentrate on S_2 . Denote the bandwidth and the extent of the instance produced by the rounding of Step 2 by β'' and R'' , respectively. Consider the rounding first. By the code, we have that for all i ,

$$\frac{1}{\sqrt{2}} \leq \frac{\tau''_i}{\tau_i} \leq \sqrt{2}. \quad (2)$$

It follows from Eq. (2) that the rounding of Step 2 contributes a factor of at most $\sqrt{2}$ to the period approximation of S_2 . By Theorem 3.1, the application of Algorithm cont_bal in Step 2 increases the period approximation of S_2 by at most another factor of $\beta'' + R''$. It is immediate from Eq. (2) that $R'' \leq \sqrt{2}R$. We now bound β'' .

Partition the set of jobs into two subsets, G_1 and G_2 , such that

$$\begin{aligned} G_1 &= \{i \mid \log \tau_i - \lfloor \log \tau_i \rfloor \leq 1/2\} \\ G_2 &= \{i \mid \log \tau_i - \lfloor \log \tau_i \rfloor > 1/2\}. \end{aligned}$$

I.e., G_1 consists of all jobs whose periods were rounded down in Step 2, and G_2 consists of all jobs whose periods were rounded up in Step 2. Denote the total requested bandwidths of G_1 and G_2 by β_1, β_2 respectively. We start by bounding β'' in terms of β_1 : this is based on the observation that in the rounding step, the bandwidth of jobs in G_1 may increase by a factor of at most $\sqrt{2}$, and the bandwidth of jobs in G_2 does not increase. Adding the fact that $\beta_2 \leq 1 - \beta_1$, we get

$$\begin{aligned} \beta'' &\leq \sqrt{2}\beta_1 + \beta_2 \\ &\leq \sqrt{2}\beta_1 + (1 - \beta_1) = 1 + (\sqrt{2} - 1)\beta_1. \end{aligned} \quad (3)$$

Next, we bound β_1 in terms of β' . Recall that β_1 is defined by the rounding of Step 1. In that rounding, the periods of jobs in G_1 are *increased* by a factor of at least $\sqrt{2}$, and the periods of jobs in G_2 are not decreased. Hence $\beta' \leq \beta_1/\sqrt{2} + \beta_2$. Since in our case, $\Delta' < \delta_0$, i.e., $1 - \beta' < \delta_0$, we can conclude that

$$\begin{aligned} \delta_0 &> 1 - \beta' \\ &\geq 1 - \left(\frac{\beta_1}{\sqrt{2}} + \beta_2 \right) \\ &\geq 1 - \left(\frac{\beta_1}{\sqrt{2}} + (1 - \beta_1) \right) \\ &= \left(1 - \frac{1}{\sqrt{2}} \right) \beta_1. \end{aligned}$$

Rearranging, we get

$$\beta_1 < \frac{\sqrt{2}\delta_0}{\sqrt{2}-1} \quad (4)$$

Combining Eqs. (3,4), with the observations above, we obtain a bound on the approximation ratio of S_2 in this case:

$$\begin{aligned} \rho(\mathcal{J}, S_2) &\leq \sqrt{2}(\beta'' + 2^{-g}R'') \\ &\leq \sqrt{2}\left(1 + (\sqrt{2}-1)\beta_1 + 2^{\frac{1}{2}-g}R\right) \\ &< \sqrt{2}\left(1 + (\sqrt{2}-1) \cdot \frac{\sqrt{2}\delta_0}{\sqrt{2}-1} + 2^{\frac{1}{2}-g}R\right) \\ &= \sqrt{2} + 2\delta_0 + 2^{1-g}R. \end{aligned} \quad (5)$$

We can now conclude the proof, using the bounds of Eq. (1) and Eq. (5).

$$\begin{aligned} \rho(\mathcal{J}, S) &\leq \max(\rho(\mathcal{J}, S_1), \rho(\mathcal{J}, S_2)) \\ &\leq \max(2-2\delta_0+R2^{1-g}, \sqrt{2}+2\delta_0+R2^{1-g}). \end{aligned} \quad (6)$$

The maximum in Eq. (6) is minimized when the two expressions of Eq. (6) are equal; trivial algebra shows that equality is obtained when $\delta_0 = \frac{1}{2} - \frac{1}{2\sqrt{2}}$, yielding $\rho(\mathcal{J}, S) \leq 1 + \frac{\sqrt{2}}{2} + 2^{1-g}R$. The jitter ratio in this case, obviously, is Rg as well. ■

5. Conclusion

In this paper we explored the idea of reducing the rate allocated to periodic tasks for the benefit of having smaller jitter. This tradeoff may be useful for mobile devices, where reduced jitter can be translated to reduced power consumption. We believe that this approach deserves further study.

The model used in this paper is the *slotted time* model, where all jobs have integer lengths and integer start times. In the *unslotted* model, job lengths, requested periods, start times (and hence granted periods as well) may be any positive real number. We remark that the algorithms presented in this paper can be extended to the unslotted version.

Acknowledgment

We wish to thank Nir Naaman and Raphael Rom for useful discussions and for providing us with a preprint of [7].

References

- [1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: data management of asymmetric communication environments. In *Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1995.
- [2] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. In *International Conference on Real-Time Computing Systems and Applications*, pages 62–69, Hong Kong, Dec. 1999. IEEE Computer Society Press.
- [3] Bluetooth technical specifications, version 1.1. Available from <http://www.bluetooth.com/>, Feb. 2001.
- [4] Z. Brakerski, A. Nisgav, and B. Patt-Shamir. General perfectly periodic scheduling. In *Proc. 21st Ann. ACM Symp. on Principles of Distributed Computing*, pages 163–172, Monterey, CA, July 2002.
- [5] M. B. Jones, D. Roşu, and M.-C. Roşu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *6th ACM Symposium on Operating Systems Principles (SOSP)*, pages 198–211, Saint-Malo, France, Oct. 1997.
- [6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.
- [7] N. Naaman and R. Rom. Scheduling real-time constant bit rate flows over a TDMA channel. Technical Report CCIT 410, Dept. of Electrical Engineering, Technion, Dec. 2002.