Algorithmica (2006) 45: 183–208
DOI: 10.1007/s00453-005-1182-x

Algorithmica

General Perfectly Periodic Scheduling[1]

Zvika Brakerski,[2] Aviv Nisgav,[2] and Boaz Patt-Shamir[2]

Abstract. In a perfectly periodic schedule, each job must be scheduled precisely every some fixed number of time units after its previous occurrence. Traditionally, motivated by centralized systems, the perfect periodicity requirement is relaxed, the main goal being to attain the requested average rate. Recently, motivated by mobile clients with limited power supply, perfect periodicity seems to be an attractive alternative that allows clients to save energy by reducing their "busy waiting" time. In this case, clients may be willing to compromise their requested service rate in order to get perfect periodicity. In this paper we study a general model of perfectly periodic schedules, where each job has a requested period and a length; we assume that m jobs can be served in parallel for some given m. Job lengths may not be truncated, but granted periods may be different than the requested periods. We present an algorithm which computes schedules such that the worst-case proportion between the requested period and the granted period is guaranteed to be close to the lower bound. This algorithm improves on previous algorithms for perfect schedules in providing a worst-case guarantee rather than an average-case guarantee, in generalizing unit length jobs to arbitrary length jobs, and in generalizing the single-server model to multiple servers.

Key Words. Periodic scheduling, Approximation algorithms.

1. Introduction.

Consider a system that comprises a resource and several clients sharing it by means of time multiplexing. In many application domains (e.g., real-time tasks, multimedia applications, communication with guaranteed quality-of-service), clients need to be scheduled periodically at some prescribed rate without preemption. Practically, this means that the time axis is divided into "slices" which are allocated to clients. The allocation of slices to clients is governed by a scheduling algorithm: given a set of jobs (corresponding to clients), each with its own *length* and *requested period*, the scheduling algorithm produces an assignment of time to clients, while trying to optimize two different measures:

- **Approximation:** a schedule is said to have good approximation if the average time between two consecutive occurrences of the same job is close to the requested period of that job, according to some given metric.
- **Smoothness:** a schedule is said to have good smoothness if the occurrences of each job are as evenly spaced as possible (under some other given metric).

The metrics may vary according to the application at hand; in any case, it is clear that the best possible approximation is achieved when the granted periods are exactly the

[1] An extended abstract of this work appeared in *Proceedings of the* 21*st Annual ACM Symposium on Principles of Distributed Computing*, pages 163–172, Monterey, California, July 2002.
[2] Department of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel. {zvika,aviv,boaz}@eng.tau.ac.il.

Received November 23, 2003; revised March 15, 2005, and June 1, 2005. Communicated by A. Borodin. Online publication March 3, 2006.

requested periods, and the best possible smoothness is achieved by *perfectly periodic* schedules, where each job is scheduled *exactly* every p time units, for some p called the *period* of that job. Note that it is easy to optimize one objective while neglecting the other: Approximation can be trivially achieved to any desired degree by taking long intervals of time and partitioning them to contiguous blocks, the lengths of which are proportional to the requested bandwidth (the bandwidth of a job is its length divided by its period), with some rounding. The longer the sequence is, the better the approximation that can be guaranteed; but clearly, the longer the sequence is, the worse the smoothness becomes. At the other end of the spectrum, we have the round-robin schedule, which disregards the requested periods. Round-robin features the best possible smoothness, but obviously suffers from poor approximation. Most prior work on periodic scheduling has concentrated on obtaining good approximation, while smoothness was relaxed in various ways. In this paper we explore the other extreme: *we insist on maintaining strict smoothness and try to achieve the best approximation under this restriction*. That is, we consider the case where the schedule must be perfectly periodic, but the granted periods may be different from the requested periods. Our goal is to optimize the approximation measure under the perfect periodicity constraint.

Perfect schedules are not always feasible if the requested periods may not be changed. Consider, for example, a slotted time model, and suppose that one client requests period 2 and another requests period 3. There is no way to satisfy both requests: the first client must occupy either all the even-numbered slots or all the odd-numbered slots, but the second client must occupy some even-numbered and some odd-numbered slots. It therefore follows that if perfect periodicity is sought, there are cases where the periods granted will not match the requests. Moreover, it is NP-hard even to decide whether a given set of requests allows a perfectly periodic schedule [6].

Despite their limitations, perfect schedules are attractive from several aspects, all arising from the fact that they are very simple to describe mathematically: the schedule of a client is completely specified by only two numbers (period and offset). This inherent simplicity gives rise to several desirable consequences; we list a few.

Wireless communication with portable devices. One of the major power consumers in portable networked devices is the transceiver, used for wireless communication. Power is a critical issue, since a portable device may weigh only few grams, leaving very little room for batteries. Perfect periodic schedules can significantly reduce the power requirement of a mobile client while it is waiting for its turn: instead of “busy waiting” (i.e., constantly listening to the radio channel), the device can turn on its radio exactly when its turn arrives. This feature exists in modern wireless technologies [13]. For example, the Bluetooth standard [9] defines *sniff mode*, in which a device listens to the network only at intervals defined in a strictly periodic fashion, allowing it to shut off its transceiver at other times. Another example is *broadcast disks* [1], where a server continuously broadcasts a “database.” A client that wishes to “read” an item in the database waits until that item is scheduled. If the schedule is perfectly periodic, the client can trivially compute the next occurrence of the desired item. Moreover, if the schedule is perfectly periodic, then “index pages” can be interleaved between the data items so that a randomly arriving client does not need to listen continuously until its desired data page is broadcasted [14], [18]. Index pages significantly reduce the active listening time of the client. No such scheme exists for non-perfect schedules.

Fairness. Another, perhaps more abstract, motivation for perfect periodicity is that in time-sharing systems, one of the main objectives of schedules is fairness: intuitively, fairness means that the number of time slots client i waits should always be proportional to its average period. A social example of the fairness requirement is the classical *chairperson assignment problem* [20], which can be illustrated with the following example. A union of several states changes its chairperson every year. The schedule should be fair: each state gets its share of chairing the union according to its size, say. However, the schedule should also attain this fairness quickly: no state would agree to wait hundreds of years to get its first term of chairing the union.

What constitutes a good solution? Several fairness criteria have been suggested. For example, in some network models, each client i has two parameters (w_i, r_i), and the requirement is that in any time window of length $T \geq w_i$, client i gets at least $\lfloor r_i T \rfloor$ time slots [10]. A stricter requirement is the *prefix criterion*, where the requirement is that in any prefix of T slots, each client i gets either $\lfloor \alpha_i T \rfloor$ or $\lceil \alpha_i T \rceil$ slots, where α_i is the share allocated to client i [19]. Since the number of slots is integral, this seems to be the best possible solution. Indeed, there exists a schedule that meets the prefix fairness requirement [20]. Still, the *gap* between two occurrences of the same client could be as large as twice its average gap. While this may be acceptable for some periodic tasks, such variability in the periodicity does not reduce the busy waiting time, thus defeating the goal of power saving in the wireless communication scenario described above.

Our results. In a sense, the goal of this work is to determine the limits of perfect periodicity. Specifically, we study perfectly periodic scheduling under the worst-case approximation measure. We assume that each job i has a requested period τ_i and a length b_i, and the requirement is that the schedule must allocate b_i time units for each occurrence of that job in a perfectly periodic fashion. The quality of the schedule is the maximum, over all jobs, of the period of the job in the schedule divided by its requested period. We start by showing that the multiple-length case is inherently different from the unit-length case (where $b_i = 1$ for all i). We prove that, in contrast to the unit-length model, even if all lengths and periods are powers of 2, there may be no perfect schedule that satisfies the requests without changing the periods. It turns out that the ratio between the largest job length and the shortest job period is a key quantity. Formally, we define the *extent* of a given set of requests J to be $R_J \stackrel{\text{def}}{=} \max\{b_i \mid i \in J\}/\min\{\tau_i \mid i \in J\}$. Our lower bound shows that in some cases the best possible average ratio—and hence, the maximum ratio—of the granted period to the requested periods cannot be better than essentially $1 + R_J$. We then proceed to develop algorithms with bounded approximation ratios, also expressed in terms of R_J. Specifically, we first describe an algorithm called s&b that guarantees an approximation factor of $1 + R_J$, assuming that the ratio between any two periods is a power of 2. Algorithm s&b is a technique that carefully "smears" the jobs as evenly as possible on the time axis; analysis yields the stated approximation ratio. We then describe an additional technique that can be applied to sets of schedules produced by Algorithm s&b: intuitively, this is a technique that combines schedules by interleaving, but still allows us to bound the approximation ratio in the combined schedule. Finally, in Section 6, we specify and analyze our general algorithm, which uses techniques developed in previous sections to guarantee an approximation factor of $1 + O(R_J^{1/3})$ for *any* instance (note that $R_J \leq 1$, so $R_J^{1/3} \geq R_J$). The latter algorithm extends to the *multiple server* model, where m jobs can be served in parallel for some given parameter m. Our algorithms also guarantee that no period is reduced too greatly. We prove that the approximation factor of any job is never smaller than $1 - O(R_J^{1/3})$.

Related work. Most previous work has concentrated on the weaker approximation measure of the *average* ratio between the granted periods and the requested periods. In the average ratio the weight of each job is its bandwidth request, defined to be its length divided by its requested period. The work most relevant to the current results is [5], where it is proved that if all jobs have a unit length, then there exist schedules that guarantee that the average approximation ratio is $1 + O(R_J^{1/3})$. These schedules use a hierarchical round-robin method called *tree scheduling* [4], [12]. The upper bounds in the current paper improve on the algorithm in [5] by bounding the worst-case ratio (rather than the average ratio), by allowing multiple lengths, and by considering multiple servers. The techniques we use here are not dependent upon [5] (the constant factors hiding in the asymptotic notation are actually smaller here).

Early work on perfectly periodic schedules was motivated by teletext systems. In [2] Ammar and Wong show that the optimal schedule for this problem is cyclic and give nearly optimal algorithms for the problem. The schedules they produce have an approximation factor of 2. Jones et al. [15] propose a scheduling algorithm for operating systems that can be shown to be perfectly periodic, and their approximation factor is 2 for general instances. Another variant of periodic scheduling is the maintenance problem [3], [21]. In [3] Anily et al. give an optimal solution for the case where there are only two jobs and give an approximation factor of 2 for the general case.

Minimizing the expected *waiting time* for broadcast disks has received much attention. This problem is equivalent to minimizing the average approximation ratio without the perfect periodicity requirement. For this setting, Bar-Noy et al. [6] give an approximation ratio of $\frac{9}{8}$. In their algorithm the gaps between consecutive occurrences of the same client can assume any of three distinct values (perfect periodicity means that exactly one value is allowed). In [18] Khanna and Zhou distinguish between *waiting time*, defined to be the total time until the client gets its requested item, and *tuning time*, defined to be the time the client is *active* while waiting (busy waiting). Using perfect schedules and an indexing scheme, they give a $1.5 + \varepsilon$ approximation to the average waiting time with a tuning time of $O(\log n)$. In [17] Kenyon et al. give a polynomial-time approximation scheme to the broadcast disk problem. The schedules they produce may not be periodic. All the broadcast disk results above assume all jobs have unit length; Kenyon and Schabanel [16] study multiple lengths and prove an approximation ratio of 3 for the average waiting time.

Non-perfect periodic scheduling with approximation ratio 1 is a classical issue in scheduling theory [19], [21], [8]. For example, Liu and Layland [19] 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. Baruah et al. [7] adopt Liu and Layland's definition for periodic scheduling, but they try to minimize *jitter*, which is a measure that quantifies the deviation from perfect scheduling.

Paper organization. The remainder of this paper is organized as follows. In Section 2 we formalize the model and introduce notation. In Section 3 we prove a lower bound on the approximation ratio. In Section 4 we present Algorithm s&b, which works if the ratio between any two periods is a power of 2. In Section 5 we present Algorithms split and splice, which can be applied to schedules produced by Algorithm s&b. Our main algorithm, called perfPeriodic, uses Algorithms s&b, split, and splice as subroutines. It is presented in Section 6, along with its extension to the multiple server case, called perfPeriodicM. Conclusions are presented in Section 7.

2. Problem Statement and Notation.

We use many quantities in our study. To aid the reader, we have summarized most of the notation used in this work in Figure 1.

Instances. An *instance* is a set of *n jobs* $J = \{j_i = (b_i : \tau_i)\}_{i=1}^{n}$, where b_i is the *size* or *length* of j_i, and τ_i is the *requested period* of j_i. We sometimes also refer to jobs as *clients*. The maximal length of a job in an instance J is denoted by $B_J \overset{\text{def}}{=} \max\{b_i \mid j_i \in J\}$. The maximal and minimal values of the requested periods in an instance J are denoted by $T_J \overset{\text{def}}{=} \max\{\tau_i \mid j_i \in J\}$ and $t_J \overset{\text{def}}{=} \min\{\tau_i \mid j_i \in J\}$. The ratio between B_J and t_J is called the *extent* of J, formally defined by $R_J \overset{\text{def}}{=} B_J/t_J$. For the single-server model, we assume that $R_J \le 1$ (otherwise, no schedule can satisfy the requests of J). We usually omit the J subscript when it is clear from the context.

The *requested bandwidth* of a job j_i is defined by $\beta_i \overset{\text{def}}{=} b_i/\tau_i$. We denote the total bandwidth of an instance (set of jobs) J by $\beta_J \overset{\text{def}}{=} \sum_{i=1}^{n} \beta_i$.

Schedules. A *schedule* S for an instance J is a set of *start time sequences* $S = \{I_1, \ldots, I_n\}$, where each start time sequence is an infinite monotonic sequence $I_i = \langle A_{i_1}, A_{i_2}, A_{i_3}, \ldots \rangle$. We say that job j_i is *scheduled* at time t if for some k, $A_{i_k} \le t < A_{i_k} + b_i$. We assume that $A_{i_{k+1}} \ge A_{i_k} + b_i$ for all i and k. A schedule is called *m-feasible* if for all t, at most m jobs are scheduled at time t. The parameter m is called the *number of servers*. Note that if $\beta_J > m$, where m is the number of servers, then there is no m-feasible schedule for the instance. The *free bandwidth* of an instance J is defined by

Instances:

- J: an instance of the problem.
- n: number of jobs (clients) in an instance.
- m: number of servers.
- j_i: the ith job in an instance.
- b_i: length (size) of j_i.
- τ_i: requested period of j_i.
- $B_J \overset{\text{def}}{=} \max\{b_i \mid j_i \in J\}$: maximal length (size) in instance J.
- $T_J \overset{\text{def}}{=} \max\{\tau_i \mid j_i \in J\}$: maximal requested period in instance J.
- $t_J \overset{\text{def}}{=} \min\{\tau_i \mid j_i \in J\}$: minimal requested period in instance J.
- $R_J \overset{\text{def}}{=} B_J/t_J$: extent of instance J.
- $\beta_i \overset{\text{def}}{=} b_i/\tau_i$: requested bandwidth of j_i.
- $\beta_J \overset{\text{def}}{=} \sum_{j_i \in J} \beta_i$: total requested bandwidth in instance J.
- $\Delta_J \overset{\text{def}}{=} m - \beta_J$: free (unrequested) bandwidth of instance J.

Schedules:

- S: a schedule.
- τ_i^S: granted period of j_i in schedule S.
- $\rho_i \overset{\text{def}}{=} \tau_i^S/\tau_i$: individual ratio of j_i in schedule S.
- $C_{\text{MAX}}(J, S) \overset{\text{def}}{=} \max\{\rho_i \mid j_i \in J\}$: MAX measure of instance J and schedule S.

Fig. 1. Glossary of notation.

$\Delta_J \stackrel{\text{def}}{=} m - \beta_J$. For most of this paper, we consider the single server case, i.e., $m = 1$. A schedule S is said to be *perfectly periodic* (or just *perfect* for short) if for each job j_i there exists a *granted period* τ_i^S such that for all $k \geq 1$, $A_{i_{k+1}} = A_{i_k} + \tau_i^S$. Note that the granted periods may be different from the requested periods, but the job lengths cannot be truncated by the schedule.

Performance measure. Given an instance J with schedule S, we define the *individual ratio* of a job j_i in S to be $\rho_i \stackrel{\text{def}}{=} \tau_i^S/\tau_i$. In this paper we evaluate the quality of S with respect to J using the worst-case individual ratio. Formally, $C_{\text{MAX}}(J, S) \stackrel{\text{def}}{=} \max\{\rho_i \mid j_i \in J\}$.

Slotted and unslotted models. The model presented above is for *unslotted* schedules, where the job lengths, period, and start times may be any positive real numbers. In *slotted* schedules all jobs have positive integer lengths, and all jobs start at integer times (and thus have integer periods). All algorithms presented in this paper have both slotted and unslotted versions.

3. A Lower Bound on the Approximation Factor.

We start with a simple result that shows that in some cases it is impossible to find a schedule where the ratio between the requested periods and the granted ones is less than $1 + R - O(1/t)$, where R is the extent of the instance. This result holds for *any* given values of B (the maximal job length) and t (the shortest requested period), provided they are larger than 1. In particular, it holds even in the special case where all job lengths and requested periods are powers of 2. This should be contrasted with a unit-length model (where the length of all jobs is one unit), in which it is trivial to (optimally) schedule a set of jobs if all requested periods are powers of 2. The schedule used in the proof is meaningless in a uniform-length model. In other words, the bad example below exposes an inherent discrepancy between the uniform-length and the multiple-length models.

To strengthen the negative result, rather than considering the worst-case ratio, we first prove the lower bound on the weaker measure of *average ratio*, defined by

$$C_{\text{AVE}}(J, S) \stackrel{\text{def}}{=} \frac{1}{\beta_J} \sum_{i=1}^{n} \beta_i \rho_i = \frac{1}{\beta_J} \sum_{i=1}^{n} \frac{b_i \cdot \tau_i^S}{\tau_i^2}.$$

Obviously, for any instance J and schedule S, we have $C_{\text{AVE}}(J, S) \leq C_{\text{MAX}}(J, S)$.

THEOREM 3.1. *For any given numbers B, $t \geq 1$, there exists an instance J with maximal job size B and minimal requested period t such that for all schedules S for J, we have $C_{\text{AVE}}(J, S) > 1 + R_J - (2 + R_J)/t$.*

PROOF. Define an instance J with $t - 1$ "short" jobs and one "long" job as follows: the short jobs have $b_i = 1$ and $\tau_i = t$ for $i = 1, \ldots, t - 1$; and the long job j_t has $b_t = B$, $\tau_t = Bt$. (We note that t and B need not be integers.) Clearly, the minimal period is t and the largest size is B. Consider any schedule S for J, and let τ_i^S denote the granted period

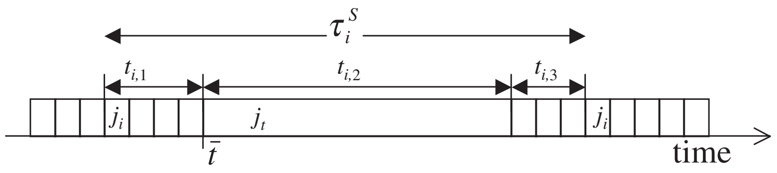

Fig. 2. Illustration of the quantities in the construction used for the lower bound proof.

of job j_i under S. Then, by definition and the construction above, and since $\beta_J = 1$, we have

$$C_{\text{AVE}}(J, S) = \sum_{i=1}^{t} \frac{b_i}{\tau_i^2} \cdot \tau_i^S = \sum_{i=1}^{t-1} \frac{1}{t^2} \cdot \tau_i^S + \frac{\tau_t^S}{Bt^2} > \frac{1}{t^2} \sum_{i=1}^{t-1} \tau_i^S. \tag{1}$$

We now bound the latter sum. Consider a time $\bar{t}$ in which the long job j_t starts, and consider, for any other job j_i, the time interval between two consecutive occurrences of j_i which contains $\bar{t}$. This interval has length τ_i^S by definition. Partition it into three parts (see Figure 2): the start of j_i until $\bar{t}$; $\bar{t}$ until the start of the job following j_t; and the start of the job following j_t until the start of j_i. Denote the lengths of these subintervals by $t_{i,1}$, $t_{i,2}$, and $t_{i,3}$, respectively. By definition, we have $\tau_i^S = t_{i,1} + t_{i,2} + t_{i,3}$ and $t_{i,2} = B$. Hence

$$\sum_{i=1}^{t-1} \tau_i^S = \sum_{i=1}^{t-1} \left(t_{i,1} + t_{i,2} + t_{i,3}\right) = \sum_{i=1}^{t-1} t_{i,1} + (t-1)B + \sum_{i=1}^{t-1} t_{i,3}. \tag{2}$$

The key observation is that $\sum_{i=1}^{t-1} t_{i,1} \geq t(t-1)/2$. This is true since the short jobs $j_1, \ldots, j_{t-1}$ must occupy at least $t - 1$ time units prior to $\bar{t}$. Similarly, $\sum_{i=1}^{t-1} t_{i,3} \geq (t-1)(t-2)/2$. Thus we get from (1) and (2) that

$$\begin{aligned} C_{\text{AVE}}(J, S) &> \frac{1}{t^2}\left(\sum_{i=1}^{t-1} t_{i,1} + (t-1)B + \sum_{i=1}^{t-1} t_{i,3}\right) \\ &\geq \frac{1}{t^2}\left(\frac{t(t-1)}{2} + (t-1)B + \frac{(t-1)(t-2)}{2}\right) \\ &= \left(1 - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right). \end{aligned}$$

□

A slightly stronger bound can be proved for the worst-case ratio.

THEOREM 3.2. *For any given numbers B, $t \geq 1$, there exists an instance J with maximal job size B and minimal requested period t such that for all schedules S for J, we have $C_{\text{MAX}}(J, S) \geq 1 + R_J - 1/t$.*

PROOF SKETCH. Use the same construction as above; observe that for the *last distinct* short job j_{i_0} scheduled after $\bar{t}$, we have $t_{i_0,3} \geq t - 2$. Since $t_{i_0,1} \geq 1$ by definition, the result follows. □

We note that the lower bounds of this section hold only for the single-server case ($m = 1$). We also note that it is straightforward to extend the latter bound to the case of any given total bandwidth $\beta > 1/t$, showing a lower bound of essentially $\beta + R$ instead of $1 + R$. We omit the (straightforward) details.

4. The Scale and Balance Algorithm.

In this section we present a basic technique for periodic scheduling with multiple lengths, which works when the ratio between any two requested periods is a power of 2. The algorithm works by spreading the jobs as evenly as possible; since perfect balancing is not always possible, some extra bandwidth is needed—either in the original instance or by scaling up the periods. The algorithms are presented in the unslotted model and then extended to the slotted model.

It has been recently brought to our attention that Algorithm bal (presented in Section 4.1 below) is similar to the algorithm proposed by Jones et al. in [15]. We note, however, that no mention of perfect periodicity is made in [15], and no analysis (or formal statement) is offered there.

4.1. *Algorithm* bal.

We start with Algorithm bal, which works if there is sufficient free bandwidth and if the ratio between any two requested periods is a power of 2. Recall that T and t denote the longest and shortest periods in the instance, respectively. The algorithm is given a parameter $t^* \le t$, such that t/t^* is a power of 2. The algorithm constructs T/t^* "bins," and distributes job start times in these bins in a balanced way. The balancing is done recursively, using a complete binary tree with T/t^* leaves. (The parameter t^*, which controls the "resolution" of the algorithm, helps to coordinate different invocations of the algorithm; the reader may find it convenient to assume $t^* = t$ for now.)

More precisely, the algorithm works as follows (see pseudocode in Figure 3 and an example of an execution in Figure 4): We associate with each node in the tree a set of "job parts," where each job part is derived from a job in the original instance, but has its own period (which may be larger than the original period). In Step 1 we associate with the root a job part for each job in the instance, where the period of the job part is exactly the requested period. The algorithm then proceeds recursively. In Step 2 the set of job parts of a node is used to create a set of job parts for each of its children; a job part with a less than maximal period is added to both children with its period doubled, and a job part with a maximal period is added to the least loaded child.

The order in which nodes are visited does not matter, but it is crucial that within a node job parts are scanned in a particular order which we denote "$\lessdot$". Specifically, $\lessdot$ orders job parts by their *original* requested period, with ties broken by index. Formally, given job parts j_i and j_k whose original requested periods are τ_i and τ_k, respectively, we say that $j_i \lessdot j_k$ if either $\tau_i < \tau_k$, or if $\tau_i = \tau_k$ and $i < k$. Note that the order between job parts depends only on their original jobs, and not on the particular job part at hand.

Finally, in Step 3, the algorithm "pads" each leaf with idle time, to make all leaves have a length of exactly t^* time units, and outputs the resulting schedule.

The main properties of Algorithm bal are summarized in the following theorem. Recall that Δ_J, the free bandwidth of instance J, is defined for the single-server case by $\Delta_J = 1 - \beta_J$ and that the extent of instance J is defined by $R_J = B_J/t_J$.

```
Algorithm bal

Input: Instance $J$, parameter $t^*$.
Output: Schedule $S$.
Code:

  1. Create a complete binary tree of $1 + \log(T/t^*)$ levels. For each job $j_i \in J$ with
     requested period $\tau_i$, add a job part with associated period $\tau_i$ to the root.
  2. Traverse the tree, starting from the root (either depth-first or breadth-first). In
     each visited non-leaf node $v$, scan all job parts of $v$ in increasing "$\lessdot$" order.
     For each scanned job part $p$, let $j(p)$ denote the original job of $p$, and let $\tau_p$
     denote the period associated with $p$:
     (a) If $\tau_p < T$, add a job part of $j(p)$ with associated period $2 \cdot \tau_p$ to each child
         of $v$.
     (b) If $\tau_p = T$, add a job part of $j(p)$ with associated period $T$ to the child of
         $v$ whose current set of job parts has less total bandwidth (the bandwidth
         of a node is the sum, over all its job parts, of the job length divided by the
         period of the job part). In case of a tie, add the job part to the left child.
  3. Scan the leaves from left to right. For each leaf $\ell$:
     (a) Output the job parts associated with $\ell$ in increasing $\lessdot$ order.
     (b) Let $w_\ell$ be the sum of lengths of job parts in $\ell$. Output $(t^* - w_\ell)$ additional
         idle time units.
```

Fig. 3. Algorithm bal.

THEOREM 4.1. *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance with free bandwidth Δ such that $\Delta \geq R(t/t^*)$, where $t^* = t/2^e$ for some integer $e \geq 0$. Suppose further that for all jobs $j_i \in J$, we have $\tau_i = 2^{e_i} t$ for some non-negative integer e_i. Then Algorithm* bal *with parameter t^* outputs a periodic schedule for J where the granted period of each job j_i is τ_i.*

Note that $\Delta \geq R(t/t^*) = B/t^*$ implies that $t^* \geq B$.

PROOF. Given any node x, let h_x denote its height (distance from the leaves), let β_x denote the total bandwidth of the job parts associated with x, and define $\Delta_x = (t^*/T)2^{h_x} - \beta_x$.

We start by proving that the algorithm is well-defined. The only problem might be in Step 3(b), so we need to prove the following:

LEMMA 4.2. *For any leaf ℓ, $\Delta_\ell \geq 0$.*

PROOF. First we claim that if y_1 and y_2 are children of an internal node x, then

$$\min(\Delta_{y_1}, \Delta_{y_2}) \geq \frac{\Delta_x - B/T}{2}. \tag{3}$$

To see that (3) is true, we assume, without loss of generality, that $\Delta_{y_1} \geq \Delta_{y_2}$. By specification of Algorithm bal, $\beta_{y_2} - \beta_{y_1} \leq B/T$, since the children may differ at most

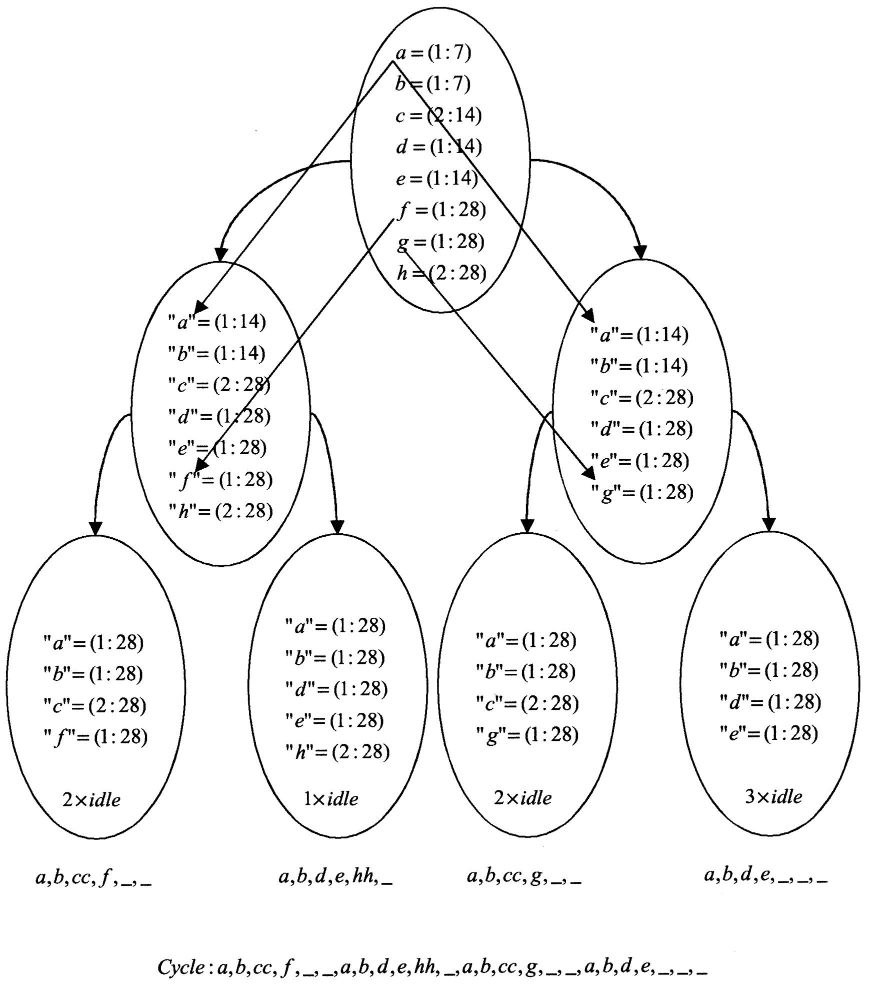

Fig. 4. Example of running Algorithm bal with parameter $t^* = t = 7$. In this instance, $\beta = \frac{5}{7}$ and $R = \frac{2}{7}$. The alphabetical order of jobs complies with their $\prec$ order. A name in quotes "x" represents a job part of original job x. Actual jobs appear only in the root of the tree. In the final cycle, a job x of length 2 is represented by xx.

by the maximal bandwidth of a single job part, whose period must be T. Hence

$$\Delta_{y_1} - \Delta_{y_2} \le \frac{B}{T}. \tag{4}$$

Also, since $\beta_x = \beta_{y_1} + \beta_{y_2}$, we know that

$$\Delta_{y_1} + \Delta_{y_2} = \Delta_x. \tag{5}$$

Combining (4) and (5) yields (3).

Now let ℓ be any leaf. We prove that $\Delta_\ell \ge 0$. Let r denote the root of the tree. For any node x, let $p^{(i)}(x)$ denote the ith ancestor of x in the tree, i.e., the node at distance

i from x on the path from x to r. Applying (3) i times, we get

$$\Delta_\ell \geq \frac{\Delta_{p^{(i)}(\ell)}}{2^i} - \frac{B}{T}\sum_{k=1}^{i}\frac{1}{2^k}. \tag{6}$$

Equation (6) holds any for $i \leq \log(T/t^*)$. Hence, using the assumption that $\Delta_r \geq R(t/t^*)$, and plugging $i = \log(T/t^*)$ in (6), we conclude that

$$\Delta_\ell > \frac{t^*}{T}\cdot\Delta_r - \frac{B}{T} \geq \frac{t^*}{T}\cdot\frac{B}{t}\cdot\frac{t}{t^*} - \frac{B}{T} = 0. \qquad \square$$

Next, we prove that Algorithm bal produces perfectly periodic schedules. For any job j_i and any node x in the tree that contains a part of j_i, define $P_x(i)$ to be the set of job parts in x that precede the part of j_i by the $\lessdot$ order. ($P_x(i)$ is uniquely defined, since a node contains at most one part of each job.) The following lemma captures the key property we need to prove perfect periodicity.

LEMMA 4.3. *Let x, y be any two nodes at the same level of the tree, such that both x and y have a job part of the same job j_i. Then $P_x(i) = P_y(i)$.*

PROOF. This property follows from the ordered and deterministic nature of Step 2 of Algorithm bal. Formally, we argue by induction on the distance d of x and y from the root r. The base case is $d = 0$, i.e., $x = y = r$ and the result is trivial. For the inductive step, suppose $d > 0$ and let x' and y' denote the parents of x and y, respectively. The inductive hypothesis is that $P_{x'}(i) = P_{y'}(i)$. Observe that the members of $P_{x'}(i)$ are distributed among the children of x' in exactly the same way they are distributed among the children of y', since the assignment of a job part to a child depends only on the job parts that precede it in the $\lessdot$ order.

If x and y are both right children or are both left children, the result follows. Otherwise assume, without loss of generality, that x is a left-child and y is a right-child and denote y's left sibling by y^*. Then there is a job part of j_i in y^* (since the dispatch scheme in x', y' is identical when j_i is scheduled); and, furthermore, $P_x(i) = P_{y^*}(i)$ by the same argument we used for the case when both are left children. Since job parts of j_i are scheduled in both children of y', then $\tau_p < T$ for all jobs in $P_{y'}(i)$ and therefore $P_y(i) = P_{y^*}(i)$, and the result follows in this case. $\square$

Clearly, Lemma 4.3 implies that the offsets of all parts of a job are the same within each node that contains them. Since the length of all nodes is equalized in the final schedule by Step 3(b), all we need to complete the proof of perfect periodicity is to argue about the set of nodes that contain the parts of a job.

LEMMA 4.4. *Assume that the nodes in a level of the tree are numbered consecutively from left to right. Suppose that a part of job j_i is in node k at level d, and that $\tau_i = T/2^{e_i}$ for some integer $e_i \geq 0$. Then:*

(1) *If $d \leq e_i$ then each node at level d contains a part of j_i.*
(2) *If $d \geq e_i$, then each node k' at level d satisfying $k' \equiv k \pmod{2^{d-e_i}}$ contains a part of j_i.*

PROOF. The first part of the lemma follows from the fact that the period associated with job parts of j_i at level $d < e_i$ is $\tau_i 2^d < T$ and therefore each of their children contains a part of j_i. For the second part, we use induction on d. The base case $d = e_i$ follows from the first part of the lemma. For the inductive step, assume that the lemma holds for level d and consider level $d + 1$. Let k_0, k_1 be the numbers of any two nodes that contain job parts of j_i at level d. Since $d \geq e_i$, the period of these job parts is T, and exactly one of the children of k_0 and k_1 will have a job part of j_i. Denote these children k'_0 and k'_1, respectively. To complete the induction step, it is sufficient to prove that $|k'_0 - k'_1| = 2|k_0 - k_1|$. To see that this is true, note that by the algorithm, the assignment of the job part of j_i to a child of k_0 and of k_1 depends only on $P_{k_0}(i)$ and $P_{k_1}(i)$, respectively. Therefore, according to Lemma 4.3, either both k'_0 and k'_1 are left children or they are both right children, and hence $|k'_0 - k'_1| = 2|k_0 - k_1|$. □

Lemmas 4.3 and 4.4 together imply that the schedules produced by Algorithm bal are perfectly periodic. Moreover, since the total time allocated for each leaf is t^* time units, Lemma 4.4 implies that the granted period of each job j_i is precisely τ_i. □

4.2. *Algorithm* s&b. Algorithm bal requires sufficient free bandwidth, which can be provided by scaling all periods appropriately. This is the only new idea in Algorithm s&b, whose pseudocode is presented in Figure 5. The only requirement left for s&b is that the periods be powers of 2 times a common factor. As with Algorithm bal, Algorithm s&b takes a parameter t^*.

THEOREM 4.5. *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance, and assume that there exists a real number $c > 0$ and some non-negative integers $e_1, \ldots, e_n$ such that $\tau_i = c \cdot 2^{e_i}$ for all i. If $t^* = t/2^e$ for some integer $e \geq 0$, then Algorithm* s&b *with parameter t^* finds a periodic schedule for J with approximation ratio $\beta + Rt/t^*$.*

Note that in Algorithm s&b, t^* is not bounded from below by B as in Algorithm bal.

PROOF. Denote $f = \beta + Rt/t^*$, and $R^* = Rt/t^*$. Then $f = \beta + R^*$, the bandwidth of the instance passed to bal is $\beta' \overset{\text{def}}{=} \beta/(\beta + R^*)$, and the extent of that instance is $R' \overset{\text{def}}{=} R/(\beta + R^*)$. Therefore, the free bandwidth of the instance submitted to bal is

$$1 - \beta' = \frac{R^*}{\beta + R^*} = R'\frac{R^*}{R} = R'\frac{t}{t^*},$$

Algorithm s&b

Input: Instance J, parameter t^*.
Output: Schedule S.
Code:

```
1. Let $f = \beta + Rt/t^*$, and let $\tau'_i = f \cdot \tau_i$ for each requested period $\tau_i$.
2. Execute Algorithm bal with requested periods $\tau'_i$ and parameter $f \cdot t^*$.
```

Fig. 5. Algorithm s&b.

and the bandwidth requirement for Algorithm bal is satisfied. Since the scaling does not change the ratios of requested periods, the result follows from Theorem 4.1 when applying Algorithm bal with argument $f \cdot t^*$. □

4.3. *Slotted Versions for Algorithms* bal *and* s&b. To derive slotted versions for Algorithms bal and s&b, all that needs to be done is to replace the padding done in Step 3(b) in Algorithm bal by an integer value. Specifically, we add $\lfloor t^* \rfloor - w_\ell$ idle time slots (instead of $t^* - w_\ell$ idle time). Recall that we require that $\Delta \geq Rt/t^*$ and therefore $t^* \geq B \geq 1$. We refer to the modified algorithm as Algorithm $\mathsf{bal}^{\mathsf{S}}$. With this slight modification, we have the following result:

THEOREM 4.6. *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance with free bandwidth Δ such that $\Delta \geq Rt/t^*$, where $R = B/t$, $t = \min\{\tau_i\}$, and $t^* = t/2^e$ for some integer $e \geq 0$. Suppose further that for all jobs $j_i \in J$, we have $\tau_i = 2^{e_i} t$ for some non-negative integer e_i. Then Algorithm $\mathsf{bal}^{\mathsf{S}}$ with parameter t^* produces a periodic schedule for J where the granted period of each job j_i is $\tau_i \lfloor t^* \rfloor / t^* \leq \tau_i$.*

The proof is identical to the proof of Theorem 4.1, with the additional observation that we can always round the schedule lengths down, since all job lengths are integral.

Using Algorithm $\mathsf{bal}^{\mathsf{S}}$ in Algorithm s&b yields a slotted version of Algorithm s&b (referred to as Algorithm $\mathsf{s\&b}^{\mathsf{S}}$), for which we have the following result:

THEOREM 4.7. *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance, and assume that there exists a real number $c > 0$ and some non-negative integers $e_1, \ldots, e_n$ such that $\tau_i = c \cdot 2^{e_i}$ for all i. If $t^* = t/2^e$ for some non-negative integer e, then Algorithm $\mathsf{s\&b}^{\mathsf{S}}$ with parameter t^* finds a periodic schedule for J with approximation ratio $\lfloor f t^* \rfloor / t^* \leq f$, where $f = \beta + Rt/t^*$.*

The proof is identical to that of Theorem 4.5, with Theorem 4.1 being replaced by Theorem 4.6.

5. Separable Schedules.

In this section we develop an additional technique we can apply to the schedules generated by Algorithm s&b. Informally, the idea is to take a few such schedules, cut them into pieces called bins, and interleave these bins in a round-robin fashion. The period of a job will be multiplied by a factor inversely proportional to the size of the bins.

It is convenient to define an abstract concept we call *Separable Schedules*. In this section we first define separable schedules, show that the schedules generated by s&b are separable, and then specify and analyze the operations split and splice that can be applied to separable schedules. We use these operations in Section 6 as subroutines.

DEFINITION 5.1. A schedule S is called *separable* if S can be partitioned into equally sized time intervals called *bins*, such that the following conditions are satisfied:

1. Each job appears at most once in each bin.

2. Each occurrence of a job that starts in some bin z ends in bin z.
3. For all jobs j_i: all bins that contain j_i have exactly the same schedule up to and including j_i.
4. If a job appears in bin k and in bin $k + l$, then it also appears in bin $k + il$ for all integers i.

The following property is a direct consequence of Properties 3 and 4:

LEMMA 5.1. *A separable schedule is perfectly periodic.*

It is straightforward to see that the schedules produced by Algorithm s&b are separable. The following lemma states the exact parameters.

LEMMA 5.2. *Let J be an instance, and let $t^* = t/2^e$ for some integer $e \geq 0$. Then Algorithm* s&b *with parameter t^* produces a separable schedule with bin size $t^*\beta + B$.*

PROOF. Consider the schedule generated by s&b. Each leaf corresponds to a bin. The bin size is therefore $f \cdot t^* = (\beta + Rt/t^*) \cdot t^* = \beta t^* + B$. Property 1 of Definition 5.1 follows from Step 2 of Algorithm bal. By construction, at most one job part of any job is distributed to each node of the tree and therefore to each leaf. Property 2 follows from Lemma 4.2. Since we can add non-negative idle time in Step 3(b), then any job that starts in some leaf ends in that leaf. Property 3 follows from Lemma 4.3. Property 4 follows from the fact that the granted periods of all jobs are a multiple of the bin size. □

5.1. *Algorithm* split. We now present Algorithm split, whose pseudocode is given in Figure 6. Algorithm split takes a separable schedule and splits each of its bins into p bins, where p is an input parameter. The split is done bin by bin, as the example in Figure 7 shows. The result is another separable schedule, as stated in the following lemma.

```
Algorithm split

Input: Separable schedule $S$ with bin size $w$, integer $p > 0$, parameter $B^* \geq B$.
Output: Schedule $S'$.
Code:

  1. Repeat for each bin $z$ of $S$:
     (a) Let current $\leftarrow 0$.
     (b) Do $p$ times:
         • Create a new bin $z'$ of $S'$. Add to $S'$, in order, all jobs whose start time
           in $z$ is between current and current $+ w/p$.
         • Let current be the finish time in $z$ of thelast job added to $z'$.
         • Add idle time units to $z'$ as needed to make its total length exactly
           $w/p + B^*$ time units.
```

Fig. 6. Algorithm split.

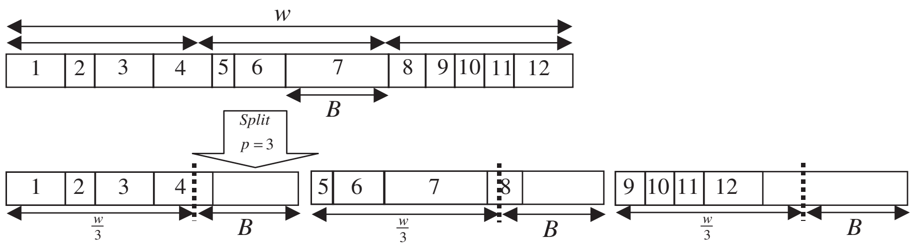

Fig. 7. Example of Algorithm split with parameters $p = 3$, $B^* = B$.

LEMMA 5.3. *Let S be a separable schedule with bin size w. Then for any given positive integer parameter p and for any parameter $B^* \geq B$, Algorithm* split *outputs a separable schedule S' with bin size $w/p + B^*$ such that $\tau_i^{S'} = (1 + pB^*/w)\tau_i^S$.*

PROOF. First we show that S' is separable. Since S is separable, no job can appear in a bin of S' more than once, and Property 1 of Definition 5.1 is satisfied. By the code of Algorithm split, each job finishes in the same bin it started in, and Property 2 is satisfied. Now, consider a job j_i in a bin z of S when z is split into bins of S'. Note that if j_i is assigned to the kth bin of S', it will be assigned to the kth bin of S' in all bins of S in which it appears, because the jobs preceding j_i in the bins of S are the same by Property 3 of S, and because the association of jobs with bins of S' is performed by order of start times. Furthermore, this also means that Property 3 holds for S'. Combined with Property 4 of S, this claim also proves Property 4 for S': since job j_i always appears in the kth bin part of its bin in S, and the bins of S it appears in are periodic, then the bins of S' where j_i appears are also periodic.

Next, we analyze the periods of jobs in S'. First, note that the period of each job in a separable schedule is a multiple of the bin length. So consider a job whose period in S is $\tau_i^S = k \cdot w$ for some positive integer k. Its period in S' is $\tau_i^{S'} = k \cdot p \cdot (w/p + B^*) = (1 + pB^*/w)\,\tau_i^S$ since each bin of size w was split into p bins of size $w/p + B^*$. □

Slotted version. Algorithm split is described above in the unslotted model. A slotted version of Algorithm split is obtained by truncating the bin size of the new schedule to $\lfloor w/p + B^* \rfloor$. The new period in the slotted model is

$$\tau_i^{S'} = \frac{p}{w}\left\lfloor \frac{w}{p} + B^* \right\rfloor \tau_i^S \leq \left(1 + \frac{pB^*}{w}\right)\tau_i^S,$$

so the approximation factor may only improve. The lemma below shows that perfect periodicity is also maintained.

LEMMA 5.4. *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance such that b_i and τ_i are integers for all i, and let S be a separable schedule with bin size w for J. Then truncating each bin of S to length $\lfloor w \rfloor$ gives a separable schedule S' with $\tau_i^{S'} = \tau_i^S \lfloor w \rfloor / w \leq \tau_i^S$ for all $j_i \in J$.*

```
Algorithm splice

Input: Separable schedules $S_1, \dots, S_k$.
Output: Merged schedule $S$.
Code:

  1. Output the round-robin schedule of bins: the first bin in $S_1$, followed by the
     first bin in $S_2$, and so on, until the first bin in $S_k$, and then the second bin in $S_1$,
     etc.
```

Fig. 8. Algorithm splice.

PROOF. From the integrality of the input and from Property 3 in Definition 5.1, it follows that truncating the bins produces a feasible schedule of J. This is because in each bin of S there are jobs of integral length followed by (possibly) idle time; truncating the bin size thus to the nearest integer keeps all jobs in the bin. The new schedule S' complies with Definition 5.1 since the bins of S' contain the same jobs as the bins of S in the same order.

From Properties 1 and 3 of Definition 5.1 it follows that τ_i^S/w is integral for any j_i. In S' the period of j_i is $\tau_i^{S'} = (\tau_i^S/w) \cdot \lfloor w \rfloor$, since the length of a bin is now $\lfloor w \rfloor$ and the offset within the bin remains unchanged. Therefore $\tau_i^{S'}/\tau_i^S = \lfloor w \rfloor / w$. □

5.2. *Algorithm* splice. Algorithm splice (whose trivial pseudocode is presented in Figure 8) merges k separable schedules into a single, perfectly periodic (but not necessarily separable) schedule. Since the input is presented as a set of separable schedules, the action of Algorithm splice is just to produce the bins of these schedules in a round-robin fashion. For perfect periodicity to be maintained, it is important that the merged schedules have disjoint job sets. The periods of jobs are multiplied by a factor inversely proportional to the size of their original bins.

LEMMA 5.5. *Let $S_1, \dots, S_k$ be separable schedules with bin sizes $w_1, \dots, w_k$, respectively. Assume that no job appears in more than one of the schedules. Then Algorithm* splice *outputs a perfectly periodic schedule S such that for all $j_i \in S_\ell$ we have $\tau_i^S = (W/w_\ell)\tau_i^{S_\ell}$, where $W = \sum_{\ell=1}^{k} w_\ell$.*

PROOF. Let j_i be some job of S_ℓ. The period of j_i in S_ℓ is $\tau_i^{S_\ell} = q \cdot w_\ell$ for some integer q. This means j_i appears every qth bin of S_ℓ, with the same offset in every bin. In the merged schedule S, a bin of j_i appears every W time units and the bins appear in order. Therefore, a bin in which j_i appears is scheduled every $q \cdot W$ time units in S, and hence $\tau_i^S = (W/w_\ell) \cdot \tau_i^{S_\ell}$. Since this holds for every job j_i, S is perfect. □

6. An Algorithm for General Instances.

In this section we present our general algorithm for the worst-case ratio measure. We first present the single server case in Section 6.2, and then explain how to generalize it to multiple servers in Section 6.2. Our bounds are expressed in terms of the requested bandwidth β and the extent R in the instance, and in terms of several parameters which we fix later.

```
Algorithm perfPeriodic

Input: Instance $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$, parameters $k, L$.
Output: Schedule $S$.
Code:

  1. Round the requested periods up to the next powers of $2^{1/k}$. Formally, let $\tau'_i \leftarrow 2^{(1/k)\lceil k \log(\tau_i)\rceil}$.
  2. Partition the jobs into $k$ classes $G_0, \ldots, G_{k-1}$ according to their $\tau'$ values: job $j_i$ is in $G_\ell$ if $\tau'_i = 2^{e+\ell/k}$ for some integer $e$.
  3. Let $t' \leftarrow \min\{\tau'_i \mid j_i \in J\}$. Let $\ell^*$ be such that $G_{\ell^*} \ni j_i$ for some job $j_i$ with $\tau'_i = t'$.
     Define $t_0 \leftarrow t' \cdot 2^{-\ell^*/k}$.
     Define $t_\ell \leftarrow t_0 \cdot 2^{\ell/k}$ for $\ell = 1, \ldots, k-1$.
  4. Apply Algorithm s&b to each class $G_\ell$ using parameter $t_\ell$. Let $S_\ell$ denote the resulting schedule for class $G_\ell$.
  5. Apply Algorithm split to each schedule $S_\ell$, using parameters $p_\ell = \lceil L \cdot 2^{\ell/k}\rceil$ $B^* = B$.
  6. Apply Algorithm splice to the $k$ schedules produced in Step 5, and output the resulting schedule.
```

Fig. 9. Algorithm perfPeriodic.

6.1. *Algorithm for the Single Server Model.* Algorithm perfPeriodic, whose pseudocode is presented in Figure 9, is stated using parameters k and L. The exact value of these parameters is determined later. Roughly speaking, the algorithm works as follows: First, the requested periods are rounded to the next power of $2^{1/k}$, thereby partitioning the jobs into k classes, where the periods differ in each class only by a factor which is a power of 2. For each class, we apply Algorithm s&b with a parameter which is essentially the smallest period in *all* classes (which is why that parameter was needed in the first place). The parameter needs to be adjusted by the appropriate multiple of $2^{1/k}$. Then each of the k schedules produced by Algorithm s&b is submitted to Algorithm split, which splits each of its bins into roughly L bins (again, adjusted by the appropriate multiple of $2^{1/k}$). Finally, all the small bins are merged back together using Algorithm splice.

Combining all the bounds and performing some algebraic manipulations, we arrive at our nearly final result. The following result not only bounds the approximation factor but in fact also bounds all individual ratios from above and below. Recall that the individual ratio for job j_i in a schedule S is defined by $\rho_i \overset{\text{def}}{=} \tau_i^S/\tau_i$.

THEOREM 6.1. *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of periodic scheduling with total requested bandwidth β_J and extent R_J. Let S be the schedule produced by Algorithm* perfPeriodic *for J with parameters k, L. Then for all $j_i \in J$,*

$$\left(1 - \frac{1}{k}\right) \cdot \left(1 - \frac{1}{L}\right) \cdot (\beta_J + k(L+1)R_J) < \rho_i$$

$$\leq \left(1 + \frac{1}{k}\right) \cdot \left(1 + \frac{1}{L}\right) \cdot (\beta_J + 2k(L+1)R_J)\,.$$

PROOF. First, we observe that the algorithm is well-defined in the sense that Algorithm s&b is applicable in Step 4. This is true since the periods of all jobs in the same G_ℓ class are powers of 2 multiplied by a common factor of $2^{\ell/k}$. In addition, the minimal period of jobs in G_ℓ is the minimal period in G_{ℓ^*} times $2^{e+(\ell-\ell^*)/k}$ for some integer $e \geq 0$, and hence t_ℓ is a power of $\frac{1}{2}$ times the minimal period in G_ℓ.

We now analyze the approximation factor step by step. In each step we give both upper and lower bounds on the individual ratios. We start with Step 1. This is an easy case, since for all jobs we clearly have

$$1 \leq \frac{\tau_i'}{\tau_i} < 2^{1/k}. \tag{7}$$

Analyzing the following steps requires us to introduce additional notation for the intermediate quantities. Consider a class G_ℓ.

- β_ℓ denotes the total bandwidth of jobs in class G_ℓ. Recall that G_ℓ is a set of *rounded* jobs and therefore $\beta_J/2^{1/k} < \sum_{\ell=0}^{k-1} \beta_\ell \leq \beta_J$.
- $r_\ell \overset{\text{def}}{=} B/t_\ell$. Note that r_ℓ is an upper bound on the extent of the jobs in G_ℓ. Note also that the same global value B is used for all ℓ.
- $f_\ell \overset{\text{def}}{=} \beta_\ell + r_\ell$.

By Theorem 4.7, Step 4 increases the periods of jobs in G_ℓ by a factor of f_ℓ. By Lemma 5.2, the bin size in S_ℓ is $t_\ell\beta_\ell + B$. Hence, by Lemma 5.3, the periods are increased in Step 5 by a factor of

$$\begin{aligned} 1 + \frac{p_\ell B}{t_\ell\beta_\ell + B} &= 1 + \frac{\lceil L2^{\ell/k} \rceil B}{f_\ell t_\ell} \\ &\leq \frac{f_\ell + (L2^{\ell/k} + 1)r_\ell}{f_\ell} \\ &= \frac{\beta_\ell + 2r_\ell + Lr_0}{f_\ell}, \end{aligned} \tag{8}$$

where the last equality follows from the fact that $r_\ell = 2^{-\ell/k} r_0$ by definition. Using the same expression, we also obtain a lower bound on the change in periods:

$$\begin{aligned} 1 + \frac{p_\ell B}{t_\ell\beta_\ell + B} &= 1 + \frac{\lceil L2^{\ell/k} \rceil B}{f_\ell t_\ell} \\ &\geq \frac{f_\ell + L2^{\ell/k} r_\ell}{f_\ell} \\ &= \frac{\beta_\ell + r_\ell + Lr_0}{f_\ell}. \end{aligned} \tag{9}$$

Consider now Step 6. Lemma 5.5 tells us how to bound the approximation factor when we are given the proportion of a bin size to the sum of all other bin sizes. Therefore, to get an upper bound on the approximation factor, we need both a lower bound on the individual bin size, and an upper bound on the total size of all bins. Let w_ℓ denote the bin size in schedule S_ℓ produced by Step 5. On the one hand, since $\lceil L2^{\ell/k} \rceil \geq L2^{\ell/k}$, we get

$$w_\ell \leq \frac{f_\ell t_\ell}{L \cdot 2^{\ell/k}} + B = \frac{t_0}{L}(\beta_\ell + r_\ell + Lr_0), \tag{10}$$

and hence the sum of the bin sizes W is at most

$$\begin{aligned} W = \sum_{\ell=0}^{k-1} w_\ell \ &\leq \ \sum_{\ell=0}^{k-1} \frac{t_0}{L}(\beta_\ell + r_\ell + Lr_0) \\ &\leq \ \frac{t_0}{L}\left(\sum_{\ell=0}^{k-1} \beta_\ell + \sum_{\ell=0}^{k-1} 2^{-\ell/k} r_0 + kLr_0\right) \\ &\leq \ \frac{t_0(\beta_J + kr_0(1+L))}{L}, \end{aligned} \tag{11}$$

since $\sum_{\ell=0}^{k-1} \beta_\ell \leq \beta_J$.

On the other hand, since $\lceil L2^{\ell/k} \rceil < L2^{\ell/k} + 1$, after some algebraic manipulation we get

$$\begin{aligned} w_\ell \ &> \ \frac{f_\ell t_\ell}{L \cdot 2^{\ell/k} + 1} + B \\ &= \ \frac{t_0(\beta_\ell + r_\ell + Lr_0(1 + 2^{-\ell/k}L^{-1}))}{L(1 + 2^{-\ell/k}L^{-1})} \\ &= \ \frac{t_0(\beta_\ell + 2r_\ell + Lr_0)}{L(1 + 2^{-\ell/k}L^{-1})}. \end{aligned} \tag{12}$$

Therefore the sum of the bin sizes W is at least

$$\begin{aligned} W = \sum_{\ell=0}^{k-1} w_\ell \ &> \ \sum_{\ell=0}^{k-1} \frac{t_0(\beta_\ell + 2r_\ell + Lr_0)}{L(1 + 2^{-\ell/k}L^{-1})} \\ &\geq \ \frac{t_0}{L(1 + L^{-1})} \sum_{\ell=0}^{k-1} (\beta_\ell + 2r_\ell + Lr_0) \\ &> \ \frac{t_0}{1+L}(\beta_J 2^{-1/k} + kr_0 + kLr_0) \\ &= \ \frac{t_0}{1+L}(\beta_J 2^{-1/k} + k(L+1)r_0) \\ &\geq \ \frac{2^{-1/k} t_0(\beta_J + 2^{1/k}k(L+1)r_0)}{1+L}, \end{aligned} \tag{13}$$

where the third inequality follows from the fact that since $\sum_{\ell=0}^{k-1} \beta_\ell > \beta_J / 2^{1/k}$.

Using (11) and (12), Lemma 5.5 says that Step 6 increases the periods by at most

$$\begin{aligned} \frac{W}{w_\ell} \ &\leq \ \frac{(t_0/L)(\beta_J + kr_0(1+L))}{(t_0(\beta_\ell + 2r_\ell + Lr_0))/(L(1 + 2^{-\ell/k}L^{-1}))} \\ &= \ \frac{(1 + 2^{-\ell/k}L^{-1})(\beta_J + kr_0(1+L))}{\beta_\ell + 2r_\ell + Lr_0}. \end{aligned} \tag{14}$$

Similarly, using (13) and (10), we obtain a lower bound on the change of periods in this step:

$$\frac{W}{w_\ell} > \frac{(2^{-1/k}t_0)(\beta_J + 2^{1/k}k(L+1)r_0)/(1+L)}{(t_0/L)(\beta_\ell + r_\ell + Lr_0)} \tag{15}$$
$$= 2^{-1/k} \cdot \frac{L}{L+1} \cdot \frac{\beta_J + 2^{1/k}k(L+1)r_0}{\beta_\ell + r_\ell + Lr_0}.$$

To get an upper bound on the overall approximation factor of Algorithm perfPeriodic, we multiply the factors in (7), (8), and (14) and another factor of f_ℓ due to Step 4, and conclude that for all $j_i \in G_\ell$, the following inequality holds:

$$\begin{aligned}\rho_i &\le 2^{1/k} \cdot f_\ell \cdot \frac{\beta_\ell + 2r_\ell + Lr_0}{f_\ell} \cdot \frac{(1 + 2^{-\ell/k}L^{-1})(\beta_J + kr_0(1+L))}{\beta_\ell + 2r_\ell + Lr_0} \\ &\le 2^{1/k}(1 + 2^{-\ell/k}L^{-1})(\beta_J + k(r_0 + Lr_0)) \\ &\le \frac{k+1}{k} \cdot \frac{L+1}{L} \cdot (\beta_J + 2kR_J(1+L)).\end{aligned}$$

The last inequality follows from the fact that $2^{1/k} \le 1 + 1/k$ for $k \ge 1$, and from the fact that $r_0 < 2R_J$ since $t_\ell > t_J/2$ for all ℓ.

Similarly, using (7), (9), and (15), we get the promised lower bound on the individual ratio:

$$\begin{aligned}\rho_i &> 1 \cdot f_\ell \cdot \frac{\beta_\ell + r_\ell + Lr_0}{f_\ell} \cdot 2^{-1/k} \cdot \frac{L}{L+1} \cdot \frac{\beta_J + 2^{1/k}k(L+1)r_0}{\beta_\ell + r_\ell + Lr_0} \\ &\ge \frac{k}{k+1} \cdot \frac{L}{L+1}(\beta_J + 2^{1/k}k(L+1)r_0) \\ &\ge \left(1 - \frac{1}{k}\right) \cdot \left(1 - \frac{1}{L}\right)(\beta_J + k(L+1)R_J),\end{aligned}$$

since $x/(x+1) \ge 1 - 1/x$ for all $x > 0$ (we use this fact for $x := k$ and $x := L$), and since $2^{1/k}r_0 \ge R_J$ by the fact that $t_0 \le 2^{1/k}t_J$. □

We note that the lower bound on the approximation factor is weaker when the total requested bandwidth β_J is small. This is because the algorithm tries to fill the schedule as if $\beta = 1$; i.e., to leave no time slot unused. If reducing granted periods too much is undesirable (say, because it increases power consumption), we can augment the original instance with dummy "place-holder" jobs with a requested period T and job length 1. After computing an approximation schedule, these dummy jobs will be replaced by idle intervals in the actual schedule. Another alternative is to multiply all granted start times by some constant factor and thus increase the approximation factor.

The following corollary shows the result of Theorem 6.1 in more concrete terms.

COROLLARY 6.2. *Let J be an instance of the periodic scheduling problem with total requested bandwidth $\beta_J = 1$ and extent R. Then there exists a schedule such that for all jobs $j_i \in J$ we have*

$$1 - 1.89R^{1/3} + O(R^{2/3}) < \rho_i < 1 + 3.78R^{1/3} + O(R^{2/3}).$$

PROOF. Apply Theorem 6.1 with parameters $k = L = (2R)^{-1/3}$. □

6.1.1. *A Slotted Version of Algorithm* perfPeriodic. To get a slotted version of Algorithm perfPeriodic, use the slotted version of Algorithm split in Step 5 of Algorithm perfPeriodic (note that we use the *unslotted* version of Algorithm s&b). In Step 6 we therefore merge schedules with bin sizes of $\lfloor w_0 \rfloor, \ldots, \lfloor w_{k-1} \rfloor$. Feasibility follows from the feasibility of Algorithm perfPeriodic and from Lemma 5.4, which guarantees the feasibility of the slotted version of Algorithm split. The approximation ratio is not increased, as we state next.

THEOREM 6.3. *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance where b_i, τ_i are integral for all i, and let S be the schedule produced by the slotted Algorithm* perfPeriodic *for J with parameters k, L. Then S is slotted and $C_{\text{MAX}}(J, S) \le (1 + 1/k) \cdot (1 + 1/L) \cdot (\beta_J + 2k(L + 1)R_J)$.*

PROOF. First we argue that we indeed get a slotted schedule: In Step 6 of the algorithm we merge schedules of integral bin size. Therefore all bins of $S_0, \ldots, S_{k-1}$ start at integral time slots. Since all job lengths are integral and the jobs of a bin are scheduled back-to-back, all jobs start at integral time slots. Therefore S is a slotted schedule.

Next we show periodicity and approximation. Note that up to Step 5, the slotted and unslotted versions of Algorithm perfPeriodic are equivalent. Let $S_0^*, \ldots, S_{k-1}^*$ denote the schedules produced in Step 5 of the unslotted version, and let S^* denote the schedule produced in Step 6 of the unslotted version. Consider a job $j_i \in G_\ell$. In Step 5, j_i is scheduled with a period of $\tau_i^{S_\ell^*}$ in the unslotted version and $\tau_i^{S_\ell} = (\tau_i^{S_\ell^*}/w_\ell) \cdot \lfloor w_\ell \rfloor$ in the slotted version. By Lemma 5.5, after Step 6, j_i has a period of $\tau_i^{S^*} = (\tau_i^{S_\ell^*}/w_\ell) \cdot W$ in the unslotted case and $\tau_i^S = (\tau_i^{S_\ell^*}/w_\ell) \cdot \sum_{\ell=0}^{k-1} \lfloor w_\ell \rfloor$ in the slotted one and therefore $\tau_i^S \le \tau_i^{S^*}$. Hence the approximation factor of the slotted version is no larger than the approximation ratio of the unslotted version. The result now follows from Theorem 6.1. □

6.2. *The Multiple Servers Case.* Using the tools we already have, it is almost straightforward to generalize Algorithm perfPeriodic above to the case of m servers. Specifically, we do it as follows. In Step 5 we split each schedule S_ℓ using parameter $p_\ell = m\lceil L2^{\ell/k} \rceil$ so that the number of bins for each class is now a multiple of m. Next, we take each "block" of m consecutive bins and distribute it over the m servers. This is possible since all m bins were split from the same bin and therefore share no common jobs.

We formalize the new Step 5 using a generalization of Algorithm split. The new algorithm, Algorithm gensplit, takes arguments m, p, B^* such that m divides p and $B^* \ge B$, and an input separable schedule S and creates m separable schedules $S_0, \ldots, S_{m-1}$. The algorithm is presented in Figure 10.

The following lemma summarizes the properties of Algorithm gensplit.

LEMMA 6.4. *Let S be a separable schedule with bin size w. Then for any given m, p, B^* such that $m|p$, $B^* \ge B$, Algorithm* gensplit *outputs separable schedules $S_0, \ldots, S_{m-1}$ with bin size $w/p + B^*$ such that $\tau_i^{S_\ell} = (1/m)(1 + pB^*/w)\tau_i^S$ for all $j_i \in S_\ell$.*

```
Algorithm gensplit

Input: Schedule $S$, parameters $m$, $p$, $B^*$ s.t. $m|p$, $B^* \geq B$.
Output: Schedules $S_0, \dots, S_{m-1}$.
Code:

  1. Apply Algorithm split with parameters $p$, $B^*$ on $S$ and enumerate the resulting
     bins in order.
  2. For each $\ell$, the output schedule $S_\ell$ is a concatenation of all bins whose index
     is congruent to $\ell$ modulo $m$.
```

Fig. 10. Algorithm gensplit .

PROOF. Let S' denote the schedule generated in Step 1 of the algorithm. By Lemma 5.3, S' is a separable schedule with bin size $w/p + B^*$ and $\tau_i^{S'} = (1 + pB^*/w)\tau_i^S$. Therefore the offset of each job j_i within a bin is constant. In order to prove the claim, all we need to show is that if j_i appears in bins numbered y and z in S', then $m|(z - y)$. This follows from Step 1(b) of Algorithm split. Assume bin z has originated from bin z^* of S and bin y originated from bin y^* of S. Clearly, $z^* \neq y^*$ since j_i cannot appear in a bin of S more than once. Since j_i appears only once in any bin of S, and since Step 1(b) of Algorithm split assigns a bin to j_i based only on its predecessors (which are identical in z^* and y^*), it follows that for some $u < p$ we have $z = p \cdot z^* + u$ and $y = p \cdot y^* + u$. Therefore $z - y = p(z^* - y^*)$ and $m|(z - y)$ since $m|p$. For every consecutive m equal sized bins in S', there is only one such bin in S_ℓ. Therefore, the periods in S_ℓ are smaller than the periods in S' by a factor of m. □

The generalized algorithm, which we call Algorithm genperfPeriodic, is identical to Algorithm perfPeriodic, except for the following modifications:

- In Step 5 apply Algorithm gensplit to each class G_ℓ with parameters m, $p_\ell = m\lceil L2^{\ell/k}\rceil$ and $B^* = B$, thus obtaining m schedules for each class. Let $S_{\ell i}$ denote the ith schedule of class G_ℓ.
- In Step 6 apply Algorithm splice m times, where the ith application merges all schedules $S_{\ell i}$, ranging over all ℓ. This produces m schedules, one for each server.

Note that Algorithm genperfPeriodic with $m = 1$ reduces to Algorithm perfPeriodic.

The approximation factor analysis is very similar to the one shown in Theorem 6.1; we have highlighted only the differences below.

THEOREM 6.5. *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance with total requested bandwidth β_J and extent R_J. Let S be the schedule for m servers produced by Algorithm* genperfPeriodic *for J with parameters k and L. Then for all $j_i \in J$,*

$$\rho_i \leq \left(1 + \frac{1}{k}\right) \cdot \left(1 + \frac{1}{L}\right) \cdot \left(\frac{\beta_J}{m} + 2k\left(L + \frac{1}{m}\right) R_J\right)$$

and

$$\rho_i > \left(1 - \frac{1}{k}\right) \cdot \left(1 - \frac{1}{L}\right) \cdot \left(\frac{\beta_J}{m} + k\left(L + \frac{1}{2}\right) R_J\right).$$

PROOF. The feasibility argument is identical to the one of Theorem 6.1 above while using Lemma 6.4 instead of Lemma 5.3. We now consider the approximation factor. Step 1 contributes a factor of at most $2^{1/k}$, and Step 4 increases the periods of jobs in G_ℓ by a factor of $f_\ell = \beta_\ell + B/t_\ell$ by Theorem 4.7. Define $r_\ell = B/t_\ell$ as above. To analyze Step 5 note that, by Lemma 6.4, the periods are increased by a factor of $1/m(1 + p_\ell B/t_\ell f_\ell) = (1/m)\,(1 + p_\ell(r_\ell/f_\ell))$, which can be bounded from above by

$$\frac{1}{m}\left(1 + p_\ell \frac{r_\ell}{f_\ell}\right) \le \frac{1}{m}\left(1 + m(L2^{\ell/k} + 1)\frac{r_\ell}{f_\ell}\right) = \frac{\beta_\ell + (m+1)r_\ell + mLr_0}{mf_\ell},$$

and from below by

$$\frac{1}{m}\left(1 + p_\ell \frac{r_\ell}{f_\ell}\right) \ge \frac{1}{m}\left(1 + mL2^{\ell/k}\frac{r_\ell}{f_\ell}\right) = \frac{\beta_\ell + r_\ell + mLr_0}{mf_\ell}.$$

To analyze Step 6 we compute the bin size w_ℓ for each S_ℓ produced by Step 5. Similarly to the single server case, we have

$$w_\ell \le \frac{f_\ell t_\ell}{mL \cdot 2^{\ell/k}} + B = \frac{t_0}{mL}(\beta_\ell + r_\ell + mLr_0),$$

and hence

$$W = \sum_{\ell=0}^{k-1} w_\ell \le \sum_{\ell=0}^{k-1} \frac{t_0}{mL}(\beta_\ell + r_\ell + mLr_0) \le \frac{t_0(\beta_J + kr_0(1 + mL))}{mL}.$$

On the other hand,

$$\begin{aligned} w_\ell &> \frac{f_\ell t_\ell}{m(L \cdot 2^{\ell/k} + 1)} + B \\ &= \frac{t_0(\beta_\ell + r_\ell + mLr_0(1 + 2^{-\ell/k}L^{-1}))}{mL(1 + 2^{-\ell/k}L^{-1})} \\ &= \frac{t_0(\beta_\ell + (m+1)r_\ell + mLr_0)}{mL(1 + 2^{-\ell/k}L^{-1})}, \end{aligned}$$

and hence

$$\begin{aligned} W = \sum_{\ell=0}^{k-1} w_\ell &> \sum_{\ell=0}^{k-1} \frac{t_0(\beta_\ell + (m+1)r_\ell + mLr_0)}{mL(1 + 2^{-\ell/k}L^{-1})} \\ &> \frac{t_0}{m(L+1)} \sum_{\ell=0}^{k-1} (\beta_\ell + (m+1)r_\ell + mLr_0) \\ &\ge \frac{t_0}{m(L+1)}\left(\beta_J 2^{-1/k} + k\frac{m+1}{2}r_0 + kmLr_0\right) \\ &\ge 2^{-1/k}\frac{t_0}{m(L+1)}(\beta_J + 2^{1/k}k(\tfrac{1}{2} + m(L + \tfrac{1}{2}))r_0). \end{aligned}$$

By Lemma 5.5, Step 6 increases the periods by W/w_ℓ, which is at most

$$\begin{aligned}\frac{W}{w_\ell} &\le \frac{(t_0/mL)(\beta_J + kr_0(1+mL))}{(t_0(\beta_\ell + (m+1)r_\ell + mLr_0))/(mL(1+2^{-\ell/k}L^{-1}))}\\ &= m\cdot\frac{(1+2^{-\ell/k}L^{-1})(\beta_J/m + kr_0(1/m+L))}{\beta_\ell + (m+1)r_\ell + mLr_0},\end{aligned}$$

and at least

$$\begin{aligned}\frac{W}{w_\ell} &> \frac{2^{-1/k}(t_0/m(L+1))\left(\beta_J + 2^{1/k}k\left(\frac{1}{2}+m(L+\frac{1}{2})\right)r_0\right)}{(t_0/mL)(\beta_\ell + r_\ell + mLr_0)}\\ &= m\cdot 2^{-1/k}\cdot\frac{L}{L+1}\cdot\frac{\beta_J/m + 2^{1/k}k\left(1/2m + L + \frac{1}{2}\right)r_0}{\beta_\ell + r_\ell + mLr_0}.\end{aligned}$$

To conclude, we multiply together all factors affecting the periods, and find that for all $j_i \in G_\ell$

$$\begin{aligned}\rho_i &\le 2^{1/k}\cdot f_\ell\cdot\frac{1}{m}\left(1+p_\ell\frac{r_\ell}{f_\ell}\right)\frac{W}{w_\ell}\\ &\le 2^{1/k}(1+2^{-\ell/k}L^{-1})\left(\frac{\beta_J}{m}+kr_0\left(\frac{1}{m}+L\right)\right)\\ &\le \frac{k+1}{k}\cdot\frac{L+1}{L}\cdot\left(\frac{\beta_J}{m}+2kR_J\left(\frac{1}{m}+L\right)\right),\end{aligned}$$

and that

$$\begin{aligned}\rho_i &> f_\ell\cdot\frac{1}{m}\left(1+p_\ell\frac{r_\ell}{f_\ell}\right)\frac{W}{w_\ell}\\ &> 2^{-1/k}\cdot\frac{L}{L+1}\cdot\left(\frac{\beta_J}{m}+2^{1/k}k\left(\frac{1}{2m}+L+\frac{1}{2}\right)r_0\right)\\ &\ge \frac{k}{k+1}\cdot\frac{L}{L+1}\cdot\left(\frac{\beta_J}{m}+k\left(L+\frac{1}{2}+\frac{1}{2m}\right)R_J\right)\\ &\ge \left(1-\frac{1}{k}\right)\cdot\left(1-\frac{1}{L}\right)\cdot\left(\frac{\beta_J}{m}+k\left(L+\frac{1}{2}\right)R_J\right).\end{aligned}$$ □

COROLLARY 6.6. *Let J be an instance of the periodic scheduling problem with total requested bandwidth $\beta_J = m$ and extent R. Then there exists a schedule for m servers such that for all jobs $j_i \in J$ we have*

$$1 - 1.89R^{1/3} + O(R^{2/3}) < \rho_i < 1 + 3.78R^{1/3} + O(R^{2/3}).$$

PROOF. Apply Algorithm genperfPeriodic with parameters $k = L = (2R)^{-1/3}$. □

7. Conclusions and Open Problems. In this paper we have studied the quality of perfect schedules in general, where jobs may have different lengths and the schedule is required to accommodate multiple servers. The quality of a schedule was evaluated using the worst-case blowup in requested periods, as opposed to the average blowup traditionally used. We have made significant progress in providing provably good algorithms, but we leave many questions open, including:

1. **Improved approximation.** We would like to achieve better approximation factors than shown above. It might be possible to compare the results of an algorithm with the optimal schedule of the given instance instead of giving an approximation ratio relative to the bandwidth.
2. **Dynamic model.** Our algorithms implicitly assume a *static model*, i.e., the set of input jobs does not change after the schedule is constructed. It is natural to consider a *dynamic model*, where jobs can be added or taken off dynamically.
3. **Dispatching.** Assume we have a perfectly periodic schedule for some given input instance. If we want to use this schedule for broadcasting purposes, our broadcast server must store some representation of the schedule and decide at each time slot which job needs to be broadcasted next. The problem of rapidly computing the next job to dispatch while minimizing the representation of the schedule on the server is known as the *dispatching problem*. A *dispatching scheme* for perfectly periodic schedules represented in special "tree schedules" is presented in [12]. We would like to construct good dispatching schemes for the schedules produced by the algorithms presented above.

We note that the s&b algorithm presented in this paper was recently extended to an algorithm that trades period approximation with smoothness [11].

Acknowledgment. We thank the anonymous reviewer whose comments prompted us to improve the statements of some of our results.

References

[1] Swarup Acharya, Rafael Alonso, Michael Franklin, and Stanley Zdonik. Broadcast disks: data management of asymmetric communication environments. In *Proceeding of the ACM SIGACT–SIGMOD Symposium on Principles of Database Systems*, 1995.

[2] Mostafa H. Ammar and Johnny W. Wong. The design of teletext broadcast cycles. *Performance Evaluation*, 5(4):235–242, Dec. 1985.

[3] Shoshana Anily, Celia A. Glass, and Rafael Hassin. Scheduling of maintenance services to three machines. *Annals of Operations Research*, 86:375–391, 1999.

[4] Amotz Bar-Noy, Vladimir Dreizin, and Boaz Patt-Shamir. Efficient periodic scheduling by trees. In *Proceedings of the* 21*st Annual Joint Conference of the IEEE Computer and Communications Societies* (*INFOCOM*), June 2002.

[5] Amotz Bar-Noy, Aviv Nisgav, and Boaz Patt-Shamir. Nearly optimal perfectly periodic schedules. *Distributed Computing*, 15(4):207–220, 2002.

[6] Amotz Bar-Noy, Bhatia Randeep, Joseph Naor, and Baruch Schieber. Minimizing service and operation cost of periodic scheduling. In *Proceedings of the* 9*th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 11–20, 1998.

[7] Sanjoy Baruah, Giorgio Buttazzo, Sergey Gorinsky, and Giuseppe Lipari. Scheduling periodic task systems to minimize output jitter. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, pages 62–69, Hong Kong, December 1999. IEEE Computer Society Press, Los Alamitos, CA.

[8] Sanjoy K. Baruah, N. K. Cohen, C. Greg Plaxton, and Donald A. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[9] Bluetooth technical specifications, version 1.1. Available from http://www.bluetooth.com/, Feb. 2001.

[10] Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, 2001.

[11] Zvika Brakerski and Boaz Patt-Shamir. Jitter-approximation tradeoff for periodic scheduling. In *Proceedings of the* 18*th International Parallel and Distributed Processing Symposium* (*IPDPS* 2004), Santa Fe, New Mexico, April 2004. IEEE Computer Society, New York.

[12] Zvika Brakerski, Vladimir Dreizin, and Boaz Patt-Shamir. Dispatching in perfectly-periodic schedules. *Journal of Algorithms*, 49(2):219–239, 2003.

[13] Yon Dohn Chung and Myoung-Ho Kim. QEM: a scheduling method for wireless broadcast data. In *Database Systems for Advanced Applications*, *Proceedings of the Sixth International Conference on Database Systems for Advanced Applications* (*DASFAA*), *April* 19–21, 1999, *Hsinchu*, *Taiwan*, pages 135–142. IEEE Computer Society, New York.

[14] Tomasz Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. In *Proceedings of the* 1994 *ACM SIGMOD International Conference on Management of Data*, *Minneapolis*, *Minnesota*, *May* 24-27, 1994, pages 25–36. ACM Press, New York.

[15] Michael B. Jones, Daniela Roşu, and Marcel-Cătălin Roşu. CPU reservations and time constraints: efficient, predictable scheduling of independent activities. In *Proceedings of the* 6*th ACM Symposium on Operating Systems Principles* (*SOSP*), pages 198–211, Saint-Malo, France, October 1997.

[16] Claire Kenyon and Nicolas Schabanel. The data broadcast problem with non-uniform transmission times. In *Proceedings of the* 10*th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 547–556, Jan. 1999.

[17] Claire Kenyon, Nicolas Schabanel, and Neal Young. Polynomial-time approximation scheme for data broadcast. In *Proceeding of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 659–666, May 2000.

[18] Sanjeev Khanna and Shiyu Zhou. On indexed data broadcast. In *Proceedings of the* 30*th Annual ACM Symposium on Theory of Computing* (*STOC* 98), New York, May 23–26 1998, pages 463–472, ACM Press, New York.

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

[20] Robert Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32:323–330, 1980.

[21] Wandi Wei and C. L. Liu. On a periodic maintenance problem. *Operations Research Letters*, 2:90–93, 1983.