TEL AVIV UNIVERSITY

THE IBY AND ALADAR FLEISCHMAN FACULTY OF ENGINEERING

Department of Electrical Engineering – Systems

General Perfectly Periodic Scheduling

Thesis submitted toward the degree of
Master of Science in Electrical Engineering
in Tel Aviv University

by

Zvika Brakerski

June 2002

TEL AVIV UNIVERSITY

THE IBY AND ALADAR FLEISCHMAN FACULTY OF ENGINEERING

Department of Electrical Engineering – Systems

General Perfectly Periodic Scheduling

Thesis submitted toward the degree of
Master of Science in Electrical Engineering
in Tel Aviv University

by

Zvika Brakerski

This research work was carried out at Tel-Aviv University
in the Department of Electrical Engineering – Systems,
Faculty of Engineering
under the supervision of

Dr. Boaz Patt-Shamir

June 2002

Acknowledgements

I wish to express my gratitude to my advisor, Dr. Boaz Patt-Shamir, for his advices and feedback. I am grateful for his guidance and support.

I wish to express my gratitude to Aviv Nisgav for his ideas and for his tremendous contribution to this research.

I am thankful to Dr. Guy Even for his help throughout the past few years.

I thank the entire staff of the department of Electrical Engineering — Systems for their companionship.

I would like to express my special gratitude to Elik and Shachar for their consideration.

Abstract

In a perfectly-periodic schedule, time is divided into time-slots, and each client is scheduled precisely every some predefined number of slots, called the period of that client. Periodic schedules are useful in wireless communication and other settings. The quality of a schedule is measured by the proportion between the requested and the granted periods: either the maximum over all jobs, or the average. There exist good scheduling algorithms for the average measure in the unit-length single-server model in which all jobs are one slot long, and at most one job is served in each time unit. In this paper we study the general model, where each job may have a different length, and m jobs can be served in parallel for some given m. We give a lower bound for this model which demonstrates the inherent difficulty of multiple lengths, and present a sequence of algorithms, culminating in an algorithm for the general case which is asymptotically optimal under the maximum ratio measure (and hence also the average ratio measure). The new algorithms utilize new techniques which are rather different from the known algorithms used for the unit-length model. Some of the algorithms improve on the best known bounds for the unit-length model.

Contents

1 Introduction **1**

2 Problem Statement and Notation **6**

3 A Lower Bound on the Approximation Factor **8**

3.1 A Lower Bound for the AVE Measure 8

3.2 A Lower Bound for the MAX Measure 11

4 The Scale & Balance Algorithm **12**

4.1 The Balance Algorithm 12

4.2 The Scale & Balance Algorithm 17

4.3 Unslotted Versions for Algorithms bal and s&b 18

4.4 A $\frac{9}{8} + O(R)$ Approximation Algorithm for AVE 19

5 A $1 + \frac{\sqrt{2}}{2} + O(R)$ Approximation Algorithm for MAX **22**

6 Separable Schedules **27**

6.1 Definition of Separable Schedules 27

6.2 Operations on Separable Schedules 28

7 A General Algorithm for MAX **32**

7.1 The Single Server Case 32

7.1.1 A Slotted Version of Algorithm C 34

7.2 The Multiple Servers Case 35

8 Conclusions and Discussion **39**

8.1 Open Problems 39

List of Figures

2.1 Glossary of notation. 7

3.1 Illustration of the construction used for the lower bound proof. 8

4.1 Algorithm bal. 13
4.2 Example of running Algorithm bal. 14
4.3 Algorithm s&b. 18
4.4 Algorithm A. 20

5.1 Algorithm B. 23
5.2 Algorithm B - the first step. 23
5.3 Algorithm B - the small Δ' case. 25

6.1 Algorithm P. 28
6.2 Example of Algorithm P. 28
6.3 Algorithm M. 30
6.4 Example of Algorithm M. 30

7.1 Algorithm C. 33
7.2 Algorithm genP. 36

8.1 Summary of presented algorithms and techniques. 40

Chapter 1

Introduction

Consider a system that comprises a resource and some clients sharing it by means of time multiplexing: in any given time, a different client may use the resource. Many application domains (e.g., real-time tasks, multimedia applications, communication with guaranteed quality-of-service) require that clients are served at some prescribed rate, and this rate should be as smooth as possible even in small time-windows. Practically, this means that the time axis is divided into small equal-size quanta called time slots, which are allocated to clients in more-or-less equally spaced intervals. The allocation of time slots to clients is governed by a scheduling algorithm. More precisely, the scheduling algorithm is given a set of *requested shares*, and its goal is to produce an assignment of time slots to clients, while trying to optimize two different measures:

- **Approximation:** a schedule is said to have good approximation if the fraction of time-slots allocated to each client (called *granted share* below) is close to the requested share of the client, according to some given metric.
- **Smoothness:** a schedule is said to have good smoothness if the time-slots allocated to each client are as evenly spaced as possible (under some other given metric).

The metrics may vary, according to the application at hand; in any case, the intuition is that the best possible approximation is when the granted rates are exactly the requested rates, and the best possible smoothness is when each client is scheduled exactly every p time slots, for some p called the *period* of that client. 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 sequences of time slots and partitioning them to intervals whose lengths are proportional to the requested shares, with some rounding. The longer the sequence is, the better approximation can be guaranteed; but clearly, the longer the sequence is, the worst smoothness we get. On the other end of the spectrum we have

the round-robin schedule, where each client gets one time slot in turn, regardless of its requested share. This scheme features the best possible smoothness, but suffers from poor approximation in general. Most prior work on this scheduling problem concentrated on the variant where the chief goal is to obtain good approximation, while smoothness was only of secondary importance. In this paper, we are interested in exploring the other extreme: *we insist on maintaining strict smoothness while relaxing the approximation requirement.* More precisely, we call a schedule *perfectly periodic*, if each client i is scheduled *exactly* every p_i time slots, for some p_i called the *period* of i. In our setting, the granted period may be different from the requested period. Our goal is to optimize the approximation measure under the perfect periodicity constraint.

Perfect schedules are attractive from a few viewpoints, all due to the fact that mathematically, they are very simple to describe: the schedule of a client is completely specified by two numbers (period and offset). This inherent simplicity gives rise to several pleasing consequences; let us list a few.

Wireless communication with portable devices. One of the major power consumers in portable devices (such as PDAs) is their radio, used for wireless communication. This is a critical issue, since a portable device may weigh only few grams, leaving very little room for batteries. Perfect schedules help to significantly reduce the power requirement of a mobile client while it is waiting for its turn: instead of "busy waiting" (constantly listening to the radio channel), the device can actually shut off its radio until its turn arrives. This feature exists in modern wireless technologies [11]. For example, in Bluetooth, which is a new technology for wireless communication of small devices, the standard defines *sniff mode* [1]. A device in sniff mode is obliged to listen to the network only in time windows defined in a strictly periodic fashion. Another example is the concept of broadcast disks [2], where the server continuously broadcasts a "database." A client that wishes to access a certain "page" in the database waits until that item is scheduled. If the schedule is perfectly periodic, it is extremely easy for the client to compute when will be the next occurrence of its desired item. Moreover, if the schedule is perfectly periodic, then it is known [12, 15] how to interleave "index pages" between the data pages so that a randomly arriving client does not need to continuously listen until its desired data page is broadcasted. The index pages reduce significantly the active listening time of the client. There is no such scheme for non-perfect schedules.

Fairness. Another important motivation for perfect periodicity is that in time-sharing systems, one of the main objectives of schedules is that they should be fair: intuitively, fairness means that the number of time slots client i waits should always be inversely proportional to its share. A social example for this requirement is the classical *chairperson assignment problem* [17], that 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 were 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 [9]. 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 [16]. Since the number of slots is integral, this seems to be the best possible. Indeed, there exists a schedule that meets the prefix fairness requirement [17]. Still, the *gap* between two occurrences of the same client could be as large as twice its average gap. When fairness is very important, perfect schedules provide the best solution.

What's known. Early work on perfectly periodic schedules was motivated by teletex systems. In [3], Ammar and Wong consider the problem of minimizing the average response time in Teletext Systems (which is equivalent to the Broadcast Disks problem presented above and further discussed below). They show that the optimal schedule for this problem is cyclic and give nearly-optimal algorithms for the problem. The schedules they produce can be easily modified to create perfectly-periodic schedules obtaining an approximation factor of 2.

Another variant is the maintenance problem [4, 19]. In [4], 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 waiting time for broadcast disks is studied in [14, 5]. Non-perfect schedules are also studied in [16, 19, 6]. In [5], Bar-Noy *et al.* give an approximation ratio of $\frac{9}{8}$ to the problem of minimizing the average waiting time in a broadcast disks setting. The algorithm uses the golden ratio schedule and therefore the gaps between consecutive runs of the same client can take any of three distinct values (for perfect schedules, only one value is allowed). The lower bound for the optimum in [5] is a relaxation of the problem to a "fluid" model where more than one job can run at each time slot. The resulting schedule in the "fluid" model is equivalent to a perfect schedule in the "non-fluid" model. Therefore approximating the perfect scheduling problem (even for the average measure) gives an approximation for the Broadcast Disks problem. In [15], Khanna and Zhou define *waiting time* as the total time until the client gets its requested share and *tuning time* as the time the client is *active* while waiting (busy waiting). They use an indexing scheme that applies only to perfect schedules to give a $1.5 + \varepsilon$ approximation to the average waiting time with tuning time of $O(\log n)$. In [14], Kenyon *et al.* give a polynomial-time approximation scheme to the Broadcast Disks

problem. The schedules they produce are not periodic in general.

The main reason to study non-perfect schedules, apparently, is that perfect schedules are not always feasible: Consider, for example, the case where 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 admits a perfectly periodic schedule [5]. Fortunately, it turns out that perfect periodicity is not necessarily expensive in terms of approximation. Specifically, define for each client its approximation ratio to be the proportion between its requested period and its granted period. It is known [10] that if all jobs have unit size, then there exist schedules that guarantee that the *average* approximation ratio (where the weight of each client is its requested bandwidth) is close to optimal. These schedules use a hierarchical round-robin method, called *tree scheduling*. In tree scheduling, the schedule is built in the form of a tree where the leaves represent the jobs. The period of a leaf is the product of the number of children of its ancestors. Tree scheduling was further investigated in [7, 8]. In [7], the *maximum* measure is also studied: under this measure, the quality of the schedule is the worst-case approximation ratio over all clients. Bar-Noy *et al.* show in [7] how to construct optimal tree schedules (which don't necessarily provide an optimal solution to the problem) in exponential-time for either the maximum or average measure.

The results above apply only to the unit-size model where all jobs have the same length. The multiple-size case for the Broadcast Disks problem has been studied in [13] where an approximation ratio of 3 was presented.

Our results and thesis organization. In this work, we extend the concept of perfectly periodic schedules in two ways. First, we consider the *multiple lengths* model, in which each client i, in addition to its requested period τ_i, also has a length b_i, and the requirement is that the schedule must allocate b_i time slots for each occurrence of that client. The occurrences must be perfectly periodic as usual. Secondly, we consider the *multiple server* model, where we assume that in each time slot, m clients can be served in parallel, and the total requested bandwidth is m. We investigate these models under both the average and the maximum measures. We start by showing that the multiple length case is inherently different from the unit-length case: 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. It turns out that the ratio between the largest job size and the shortest period is a key quantity. Formally, we define the *extent* of a given

instance J to be $R_J \stackrel{\text{def}}{=} \frac{\max\{b_i | i \in J\}}{\min\{\tau_i | i \in J\}}$. Our lower bound shows that in general, the best possible average ratio (and hence, maximum ratio) is at least $1+R_J-O(1/\min\{\tau_i \mid i \in J\})$. This lower bound is presented in Chapter 3. We then proceed to provide upper bounds on the approximation ratio, also expressed in terms of R_J. The algorithms are presented in a succession of refinements. In Chapter 4 we give an algorithm that guarantees a good upper bound for the MAX measure, provided that all periods are powers of 2 times a common multiple. Based on this algorithm, we analyze a simple algorithm that gets an approximation ratio of $\frac{9}{8} + \frac{3}{2}R_J + \frac{1}{2}R_J^2$ for the AVE measure. In Chapter 5 we present another algorithm that is based on the basic algorithm of Chapter 4. This algorithm guarantees an approximation ratio of at most $1+\sqrt{2}/2+2R \approx 1.707+2R$ for the MAX measure. A generalization of the schedules produced by the algorithm of Chapter 4 is presented in Chapter 6. This generalization is used to achieve maximum ratio guarantee of $1 + O(R_J^{1/3})$, in the algorithm presented in Chapter 7. This algorithm easily generalizes to the multiple server case. In Chapter 8 we conclude and summarize all of the algorithms presented (in Figure 8.1) and discuss relevant open problems.

The formal model is presented in Chapter 2.

Chapter 2

Problem Statement and Notation

Most of the notation used in this work appears in the Glossary in Figure 2.1.

Instances. An instance of the perfectly-periodic scheduling problem 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 refer to jobs also as *clients*. The maximal length of a job in an instance J is defined by $B_J \stackrel{\text{def}}{=} \max\{b_i \mid i \in J\}$. The maximal and minimal values of the requested periods in instance J are defined by $T_J \stackrel{\text{def}}{=} \max\{\tau_i \mid i \in J\}$, and $t_J \stackrel{\text{def}}{=} \min\{\tau_i \mid i \in J\}$. The ratio between B_J and t_J is called the *extent* of J, formally defined by $R_J \stackrel{\text{def}}{=} \frac{B_J}{t_J}$. For the single-server model, we assume that $R_J \leq 1$ (otherwise, no schedule can satisfy the requests of J). We omit subscripts when they are clear from the context.

Schedules. A *schedule* S for an instance J is a set of infinite sequences of *start times* $S = \{I_1, \ldots, I_n\}$, such that $I_i = \langle A_{i_1}, A_{i_2}, A_{i_3}, \ldots\rangle$ where A_{i_k} is a nonnegative integer for each i, k. We say that job i is *scheduled* in *time slot* t if for some k, $A_{i_k} \leq t < A_{i_k} + b_i$. We assume that $A_{i_{k+1}} \geq A_{i_k} + b_i$ for all i, k. A schedule is *m-feasible* if for all t, at most m jobs are scheduled at t. The parameter m is called the *number of servers*. For the most part of this paper, we will be interested in the single server case, i.e., $m = 1$. A schedule S is said to be *perfectly periodic* (or just *periodic* for short) if for each job i there exists a *granted period* τ_i^S such that for all j, $A_{i_{j+1}} - A_{i_j} = \tau_i^S$. Note that the granted periods may be different from the requested periods, but the job lengths cannot be truncated by the schedule. Periodic schedules can be represented by their *cycles*, i.e, a sequence of time slots whose length is the least common multiple of job periods.

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 J is defined by $\beta_J \stackrel{\text{def}}{=} \sum_{i=1}^{n} \beta_i$. Note that if $\beta_J > m$ where m is the number of servers, there is no feasible schedule for the instance. The *free bandwidth* of an instance J is defined by $\Delta_J \stackrel{\text{def}}{=} m - \beta_J$. We assume that input instances for the problem have no

Instances and jobs:

- 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.
- $B_J \stackrel{\text{def}}{=} \max\{b_i \mid i \in J\}$: maximal length (size) in instance J.
- $T_J \stackrel{\text{def}}{=} \max\{\tau_i \mid i \in J\}$: maximal requested period in instance J.
- $t_J \stackrel{\text{def}}{=} \min\{\tau_i \mid i \in J\}$: minimal requested period in instance J.
- $R_J \stackrel{\text{def}}{=} \frac{B_J}{t_J}$: extent of instance J.

Schedules and quality measures:

- S: a schedule.
- τ_i^S: granted period of j_i in schedule S.
- $\beta_i \stackrel{\text{def}}{=} \frac{b_i}{\tau_i}$: requested bandwidth of j_i.
- $\beta_J \stackrel{\text{def}}{=} \sum_{i \in J} \beta_i$: total bandwidth of instance J.
- $\rho_i \stackrel{\text{def}}{=} \frac{\tau_i^S}{\tau_i}$: individual ratio of j_i in schedule S.
- $C_{MAX}(J,S) \stackrel{\text{def}}{=} \max\{\rho_i \mid i \in J\}$: MAX measure of instance J and schedule S.
- $C_{AVE}(J,S) \stackrel{\text{def}}{=} \frac{1}{\beta_J} \sum_{i \in J} \beta_i \rho_i$: AVE measure of instance J and schedule S.

Figure 2.1: Glossary of notation.

free bandwidth.

Quality measures. Let J be an instance for the perfectly periodic scheduling problem and let S be a schedule for J. The *individual ratio* of a job i in S is defined by $\rho_i \stackrel{\text{def}}{=} \frac{\tau_i^S}{\tau_i}$. The MAX measure is simply the maximum over the individual ratios, defined formally by $C_{\text{MAX}}(J,S) \stackrel{\text{def}}{=} \max\{\rho_i \mid i \in J\}$. The AVE measure is the weighted average of the individual ratios, where the weight of i is its requested bandwidth.

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

Slotted vs. unslotted. So far we have presented the model for *slotted* schedules. In such schedules, all jobs have integer lengths and in addition, in the resulting schedule, all jobs should start (and therefore end) at the beginning of a time slot (that is at an integer time). However, sometimes we would like to refer to an *unslotted* model in which the start time of a job can be any real number and so can the periods and the lengths of the jobs. All algorithms presented have both slotted and unslotted versions.

Chapter 3

A Lower Bound on the Approximation Factor

In this chapter we show that in general, it is impossible to achieve better results than $1+R-O(\frac{1}{t})$ for either the AVE or MAX measures, where R is the extent of the instance. This result is interesting because it holds for *any* given values of B (the maximal client size) and t (the shortest requested period). In particular, it holds even in the case where all job sizes and requested periods are powers of 2, which is a trivial case to schedule in the unit length model. In other words, the bad example below exposes an inherent discrepancy between the unit length and the multiple length models. On the other hand, it extends known lower bounds for the unit-length model [10].

3.1 A Lower Bound for the AVE Measure

Theorem 3.1 *For any given integers $B < t$, 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 $C_{\mathrm{AVE}}(J,S) > 1 + R_J - \frac{2+R_J}{t}$.*

Proof: Intuitively, the proof takes advantage of the variable-length property of the problem. Consider an instance containing one very long job and many short jobs with

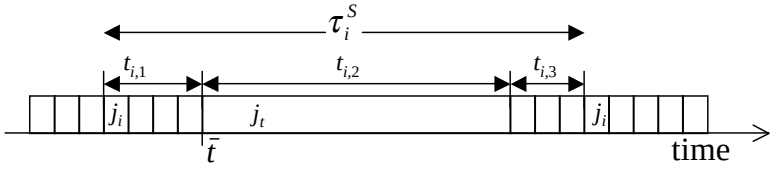


Figure 3.1: Illustration of the construction used for the lower bound proof.

short periods and consider a time where the long job is scheduled. Since the schedule is periodic, the short jobs need to be scheduled on both ends of the long job, at the shortest possible distance. Since we have many short jobs, "traffic jams" are created at both ends of the long job and a bound on the quality of the schedule is obtained.

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 $b_t = B$, $\tau_t = Bt$. 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 of job j_i under S. Then, by definition and the construction above, we have

$$\begin{aligned} C_{\text{AVE}}(J,S) &= \sum_{i=1}^{t} \frac{b_i}{\tau_i^2} \cdot \tau_i^S \\ &= \sum_{i=1}^{t-1} \frac{1}{t^2} \cdot \tau_i^S + \frac{\tau_t^S}{Bt^2} \\ &> \frac{1}{t^2} \sum_{i=1}^{t-1} \tau_i^S \ . \end{aligned} \tag{3.1}$$

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 that contains $\bar{t}$. This interval has length τ_i^S by definition; partition it into three parts (see Figure 3.1): 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 sub-intervals by $t_{i,1}, t_{i,2}$ and $t_{i,3}$, respectively. By definition, we have that $\tau_i^S = t_{i,1} + t_{i,2} + t_{i,3}$, and that $t_{i,2} = B$. Hence

$$\begin{aligned} \sum_{i=1}^{t-1} \tau_i^S &= \sum_{i=1}^{t-1} (t_{i,1} + t_{i,2} + t_{i,3}) \\ &= \sum_{i=1}^{t-1} t_{i,1} + (t-1)B + \sum_{i=1}^{t-1} t_{i,3} \ . \end{aligned} \tag{3.2}$$

The crucial observation is that $\sum_{i=1}^{t-1} t_{i,1} \geq \frac{t(t-1)}{2}$: this is true since the short jobs $j_1, \ldots, j_{t-1}$ must occupy $t-1$ distinct slots prior to $\bar{t}$. Similarly, $\sum_{i=1}^{t-1} t_{i,3} \geq \frac{(t-1)(t-2)}{2}$. Thus we get from Eq. (3.1) and Eq. (3.2) that

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

A more careful analysis gives a tighter version of the bound as shown below.

Theorem 3.2 *For any given integers $B < t$, 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 $C_{AVE}(J,S) \geq \left(1 + \frac{1}{Bt} - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right)$. Furthermore, there exists a schedule S_0 such that $C_{AVE}(J,S_0) = \left(1 + \frac{1}{Bt} - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right)$.*

Proof: The instance J is defined as in the proof of Theorem 3.1. We have

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

In the proof of Theorem 3.1 we showed that $\frac{1}{t^2}\sum_{i=1}^{t-1} \tau_i^S \geq \frac{t-1}{t}\left(1 + \frac{B}{t} - \frac{1}{t}\right)$. Now we analyze for two cases of τ_t^S in order to achieve a tight bound.

Case 1: If $\tau_t^S \geq B + (t-1)$ then

$$\begin{aligned} C_{AVE}(J,S) &= \frac{1}{t^2}\sum_{i=1}^{t-1} \tau_i^S + \frac{\tau_t^S}{Bt^2} \\ &\geq \frac{t-1}{t}\left(1 + \frac{B}{t} - \frac{1}{t}\right) + \frac{B + (t-1)}{Bt^2} \\ &= \left(1 - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right) + \frac{1}{Bt}\left(1 + \frac{B}{t} - \frac{1}{t}\right) \\ &= \left(1 + \frac{1}{Bt} - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right) \; . \end{aligned}$$

Equality holds when $\tau_i^{S_0} = B + (t-1)$ for all i, that is a round-robin scheduling of all t jobs. Therefore S_0 is the round-robin schedule.

Case 2: If $\tau_t^S < B + (t-1)$. Let $\tau_t^S = B + (t-1) - k$ for some integer $k \geq 1$ then at least k jobs have a $t_{i,3}$ value of more than B which we need to account for separately. We get

$$\begin{aligned} C_{AVE}(J,S) &= \frac{1}{t^2}\sum_{i=1}^{t-1} \tau_i^S + \frac{\tau_t^S}{Bt^2} \\ &\geq \frac{1}{t^2}\left((t-1)\cdot B + (t-1)^2 + k \cdot B\right) + \frac{B + (t-1) - k}{Bt^2} \\ &= \left(1 + \frac{1}{Bt} - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right) + \frac{1}{t^2}\left(k \cdot B - k \cdot \frac{1}{B}\right) \\ &= \left(1 + \frac{1}{Bt} - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right) + \frac{k}{t^2}\left(B - \frac{1}{B}\right) \\ &\geq \left(1 + \frac{1}{Bt} - \frac{1}{t}\right)\left(1 + \frac{B}{t} - \frac{1}{t}\right) \; . \end{aligned}$$

Since $B - \frac{1}{B} \geq 0$ for all $B \geq 1$. ∎

3.2 A Lower Bound for the MAX Measure

A slightly stronger bound can be proven in this case.

Theorem 3.3 *For any given integers $B < t$, 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 $C_{\mathrm{MAX}}(J,S) \geq 1+R-\frac{1}{t}$. Furthermore, there exists a schedule S_0 such that $C_{MAX}(J,S_0) = 1+R-\frac{1}{t}$.*

Proof: Use the same construction as in Theorem 3.1; consider the job j_{i_0} whose value of $t_{i_0,3}$ is maximal. This value is at least $t-2$ since there are at least $t-2$ jobs scheduled prior to j_{i_0} (otherwise it wouldn't have a maximal $t_{i_0,3}$). Since $t_{i_0,1} \geq 1$, we get $\tau_i^S \geq 1+B+t-2 = t+B-1$. Therefore $C_{MAX}(J,S) \geq 1+R-\frac{1}{t}$.

Equality holds when for all i, $\tau_i^{S_0} = B+(t-1)$, namely a round-robin scheduling of all t jobs. Therefore S_0 is the round-robin schedule. ∎

Chapter 4

The Scale & Balance Algorithm

In this chapter we present a basic technique for periodic scheduling with multiple lengths. The algorithm scales up the requested periods so as to have sufficient free bandwidth, and then allocates the time slots in a balanced way.

The algorithms are presented for the slotted model and then extended to the unslotted model.

The algorithms presented in this section use binary trees to construct schedules. The use of trees here, however, is quite different from "tree scheduling" presented in previous work [10, 8]. In "tree scheduling" the tree is used to construct a hierarchial round-robin schedule where in this work the tree is used merely to help the balancing process (described below).

4.1 The Balance Algorithm

We first present Algorithm bal (see Figure 4.1), that assumes that there is sufficient free bandwidth, and finds the schedule with at most the requested periods. This algorithm works when all requested periods are powers of 2 times a common constant (not necessarily integral), and it is given a parameter t^* (not necessarily integral) that is a power of $\frac{1}{2}$ times the minimal requested period t. (The parameter t^* will come into play later; for the time being, it may be convenient to assume that $t^* = t$.) The algorithm constructs a complete binary tree with T/t^* leaves (recall T is the largest requested period). Each leaf represents a run of $w = \lfloor t^* \rfloor$ time slots. The idea in the algorithm is to spread the occurrences of jobs so as not to overload the leaves. To this end, we define a total order on jobs: for jobs j_i, j_k with requested periods 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 $i < k$. The tree is constructed recursively, and

Algorithm bal

Input: Instance J, parameter t^*.

Output: Schedule S.

Code:

(1) Create a complete binary tree of $1 + \log \frac{T}{t^*}$ levels. Associate all jobs of the given instance with 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 associated with v in increasing "$\preccurlyeq$" order. For each scanned job-part j, let τ_j denote the period associated with j:

- If $\tau_j < T$, add j to both children of v, with associated periods $2 \cdot \tau_j$.
- If $\tau_j = T$, add j to the child of v whose current total associated bandwidth (i.e., sum of the job lengths divided by the associated periods) is smaller. In case of a tie, add j to the left child.

(3) Scan the leaves left-to-right. For each leaf ℓ:

3a: Output the job parts associated with ℓ in increasing $\prec$ order.

3b: Let the number of time slots used by ℓ be denoted by s_ℓ. Add $\lfloor t^* \rfloor - s_\ell$ following idle time slots.

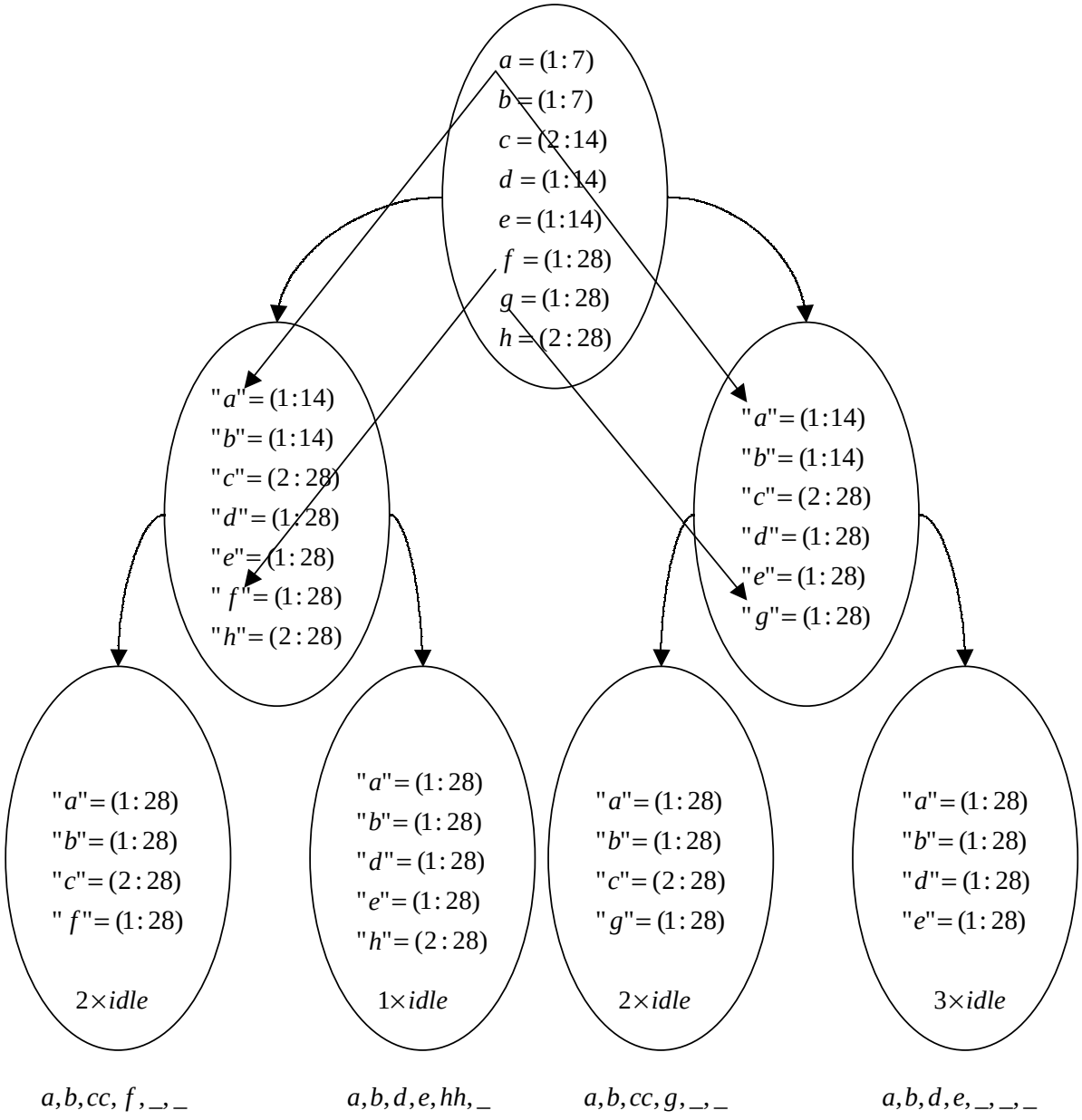
Figure 4.1: Algorithm bal.

each node will contain "job parts": each job part has its own local associated period. For the "$\preccurlyeq$" relation, each job-part uses its inherited position in "$\preccurlyeq$". The "$\preccurlyeq$" relation is determined *once* in the algorithm, before it begins, and is not re-calculated in each node, even when the job part in that node has different period than the job it originated from. See Figure 4.2 for example of how the algorithm runs.

Let us briefly justify Step 3b of Algorithm bal. We will show that given sufficient free bandwidth in the instance, the total bandwidth associated with each leaf is at most t^*/T; since the associated period of all job parts in the leaves is T, it follows that the sum of lengths of all job parts in any leaf is at most t^*. Since the lengths of the jobs are integral, the sum of sizes is at most $\lfloor t^* \rfloor$, and hence Step 3b cannot fail. We remark that completing the number of slots to $\lfloor t^* \rfloor$ is essential to the periodicity of the resulting schedule.

The main properties of Algorithm bal are summarized in the following lemma.

Lemma 4.1 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of the periodic scheduling problem with free bandwidth $\Delta \geq R\frac{t}{t^*}$, and suppose that there exists a real number $c > 0$ and nonnegative integers $e, e_1, \ldots, e_n$ such that $\tau_i = c \cdot 2^{e_i}$ for all i. If $t^* = t/2^e$, then Algorithm* bal *with parameter t^* finds a periodic schedule for J where the granted period*



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

Figure 4.2: Example of running Algorithm bal with parameter $t^* = t = 7$. The alphabetical order of jobs complies with their $\prec$ order. Note that in the root we have $\beta = \frac{2}{7}$ and $R = \frac{5}{7}$. Also note that job parts with periods smaller than T are associated with both children and jobs with period T are associated with one child only.

of each job j_i is $\tau_i \frac{\lfloor t^* \rfloor}{t^*} \leq \tau_i$.

Proof: Feasibility follows from Lemma 4.2, and periodicity and correctness are proven in Corollary 4.6. ∎

Lemma 4.2 *If $\Delta \geq R\frac{t}{t^*}$, then the bandwidth associated with each leaf by Algorithm* bal *is at most t^*/T.*

Proof: Given any node x, let h_x denote its height (distance from the leaves), let β_x denote the total bandwidth associated with x, and define $\Delta_x = \frac{t^*}{T} 2^{h_x} - \beta_x$. We need to show that $\Delta_\ell \geq 0$ for any leaf ℓ. We first claim that if y_1 and y_2 are children of an internal node x, then

$$\min(\Delta_{y_1}, \Delta_{y_2}) \geq \frac{\Delta_x - \frac{B}{T}}{2} \ . \tag{4.1}$$

To see that Eq. (4.1) is true let us assume, without loss of generality, that $\Delta_{y_1} \geq \Delta_{y_2}$. By the algorithm, $\beta_{y_2} - \beta_{y_1} \leq \frac{B}{T}$, since the children may differ at most by the maximal bandwidth of a single element. Hence

$$\Delta_{y_1} - \Delta_{y_2} \leq \frac{B}{T} \ . \tag{4.2}$$

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

$$\Delta_{y_1} + \Delta_{y_2} = \Delta_x \ . \tag{4.3}$$

Combining Eq. (4.2) and Eq. (4.3) yields Eq. (4.1). Now let ℓ be any leaf. Let $p(\cdot)$ denote the "parent of" function of the tree, and let r denote the root node. Using Eq. (4.1), and the assumption that $\Delta_r \geq R\frac{t}{t^*}$, we can conclude that

$$\begin{aligned}
\Delta_\ell &\geq \frac{\Delta_{p(\ell)}}{2} - \frac{1}{2}\frac{B}{T} \\
&\geq \frac{\Delta_{p(p(\ell))}}{4} - \frac{B}{T}\left(\frac{1}{2} + \frac{1}{4}\right) \\
&\geq \cdots \\
&> \frac{t^*}{T} \cdot \Delta_r - \frac{B}{T} \\
&\geq \frac{t^*}{T} \cdot \frac{B}{t} \cdot \frac{t}{t^*} - \frac{B}{T} \\
&= 0 \ . \quad ∎
\end{aligned}$$

Associate a schedule with each node x in the tree by applying Step 3 of Algorithm bal only to the subtree rooted at x. The following lemmas conclude the correctness of Algorithm bal.

Let x be a node in the tree and define $P_i^x \stackrel{\text{def}}{=} \{j_k \in x \mid j_k \lessdot j_i\}$ the set of job parts in x that are smaller than j_i by $\lessdot$. Denote the schedule associated with x by S_x and denote the offset of j_i in S_x by o_i^x.

Lemma 4.3 *Let x, y be two nodes of the same level in the tree, both containing job part j_i. Then $P_i^x = P_i^y$.*

Proof: This property follows from the in-order and deterministic nature of step 2 of Algorithm bal. We prove by induction on ℓ, the level of x, y. That is, their distance from the root. Let $p(\cdot)$ denote the "parent of" function of the tree.

Base: If $\ell = 0$ then $x = y$ and they are both the root of the tree and the result is trivial.

Induction: Consider the status of $p(x), p(y)$ when step 2 of Algorithm bal is executed, right before j_i is split between the children. By induction, $P_i^{p(x)} = P_i^{p(y)}$. If in $p(x), p(y)$ we have $\tau_i < T$ then j_i and all of its predecessors by $\lessdot$ are split to both children and therefore $P_i^x = P_i^{p(x)} = P_i^{p(y)} = P_i^y$. Otherwise, since the algorithm runs in order and deterministically, the status of the children of $p(x), p(y)$ is identical. Due to determinism, j_i is treated in the same way in $p(x), p(y)$ and therefore $P_i^x = P_i^y$. ∎

Lemma 4.4 *There exists a function φ that computes o_i^x from P_i^x and j_i, namely: $o_i = \varphi(j_i, P_i^x)$.*

Proof: We prove by induction on the height of x, h_x.

Base: Here x is a leaf and therefore $h_x = 0$. In this case $o_i^x = \varphi(j_i, P_i^x) = \sum_{j_k \in P_i^x} b_k$ according to step 3a of Algorithm bal.

Induction: Let y_l, y_r denote x's left and right children respectively.

- If $\tau_i = T$ then j_i is scheduled only in S_{y_l} or only in S_{y_r}. The child with which j_i is associated is a function of P_i^x since Algorithm bal is performed in-order on each node. Therefore, by induction, in this case

$$o_i^x = \varphi(j_i, P_i^x) = \begin{cases} \varphi(j_i, P_i^{y_l}) & \text{if } j_i \in y_l, \\ 2^h \cdot w + \varphi(j_i, P_i^{y_r}) & \text{if } j_i \in y_r. \end{cases}$$

  Where w is the length of a leaf.

- If $\tau_i < T$ then j_i is scheduled in both S_{y_l} and S_{y_r}. Therefore in this case $o_i^x = o_i^{y_l} = \varphi(j_i, P_i^{y_l})$. Since $P_i^{y_l}$ is a function of P_i^x due to the in-order and deterministic nature of Algorithm bal, we have $\varphi(j_i, P_i^x) = \varphi(j_i, P_i^{y_l})$ in this case.

And the claim holds for all h_x. ∎

Lemma 4.5 *Let j_i be a job part at node x. Then j_i is scheduled in a perfectly periodic manner in S_x and has a period of $\tau_i^{S_x} = \tau_i \frac{w}{T} 2^h$ where w is the length of a leaf.*

Proof: We prove by induction on the height of x, h_x.

Base: Here x is a leaf and $h_x = 0$. Therefore $\tau_i = T$ and j_i is scheduled exactly once. This means, by definition, that j_i is scheduled in a perfectly periodic manner with period $\tau_i^S = w = \tau_i \frac{w}{T} 2^0$.

Induction: Let y_l, y_r denote x's left and right children respectively.

- If $\tau_i = T$ then j_i is scheduled only in S_{y_l} or only in S_{y_r}. Therefore it is scheduled in a perfectly periodic manner with period $\tau_i^{S_x} = w \cdot 2^h = \tau_i \frac{w}{T} 2^h$.
- If $\tau_i < T$ then j_i is scheduled in both S_{y_l} and S_{y_r}. In this case $P_i^x = P_i^{y_l} = P_i^{y_r}$ since all of j_i's predecessors also have $\tau < T$ (see Lemma 4.3). Therefore, by induction and by Lemma 4.4 $o_i^{y_l} = o_i^{y_r}$. From step 2 of Algorithm bal, the job parts of j_i in y_l, y_r have τ value of $2\tau_i$ and therefore, by induction $\tau_i^{S_l} = \tau_i^{S_r} = 2\tau_i \frac{w}{T} 2^{h-1} = \tau_i \frac{w}{T} 2^h$. Since $\tau_i^{S_{y_l}} = \tau_i^{S_{y_r}}$ divides the lengths of S_{y_l} and S_{y_r} (which are $w \cdot 2^h$) and since the offsets are equal, concatenating these schedules produces a perfectly periodic schedule with the same period.

And the claim holds for all h_x. ∎

Corollary 4.6 *For each job j_i, Algorithm* bal *produces a perfectly periodic schedule with period $\tau_i \frac{\lfloor t^* \rfloor}{t^*}$.*

Note that since $t^* | t$ and for all i, $t | \tau_i$, we get integral periods for all jobs.
Proof: Apply Lemma 4.5 on the root whose level is $h_r = \log \frac{T}{t^*}$ and $w = \lfloor t^* \rfloor$. ∎

4.2 The Scale & Balance Algorithm

We can now state Algorithm s&b (see Figure 4.3). This algorithm works for jobs whose periods are powers of 2 times a common factor. It gets a parameter t^* for the balancing part. Let J be an instance of the periodic scheduling problem.

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

Proof: Let $R^* = R \frac{t}{t^*}$. The bandwidth of the instance submitted to bal is $\beta' = \frac{\beta}{f} = \frac{\beta}{\beta + R^*}$ and the new extent is $R' = \frac{R}{f} = \frac{R}{\beta + R^*}$. Therefore, the free bandwidth of the instance

Algorithm s&b

Input: Instance J, parameter t^*.

Output: Schedule S.

Code:

(1) Let $f = \beta + R\frac{t}{t^*}$, and let $\tau_i' = f \cdot \tau_i$ for each requested period τ_i.

(2) Run Algorithm bal with requested periods τ_i' and parameter $f \cdot t^*$.

Figure 4.3: Algorithm s&b.

submitted to bal is $1 - \beta' = \frac{R^*}{\beta + R^*} = R' \frac{ft}{ft^*} = R' \frac{t}{t^*}$ and the bandwidth requirement holds. The scaling keeps all periods in the form of a constant factor times a power of 2, and so the result follows from Lemma 4.1 when applying Algorithm bal with argument $f \cdot t^*$. ∎

4.3 Unslotted Versions for Algorithms bal and s&b

Algorithms bal and s&b are presented above in the slotted model. To derive unslotted versions of these algorithms, all we have to do is change step 3b of Algorithm bal to add $t^* - s_\ell$ following idle time slots (a non-integral number of slots) instead of $\lfloor t^* \rfloor - s_\ell$. For this algorithm we have the following result.

Lemma 4.8 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of the periodic scheduling problem with free bandwidth $\Delta \geq R\frac{t}{t^*}$, and suppose that there exists a real number $c > 0$ and nonnegative integers $e, e_1, \ldots, e_n$ such that $\tau_i = c \cdot 2^{e_i}$ for all i. If $t^* = t/2^e$, then the unslotted Algorithm* bal *with parameter t^* finds a periodic schedule for J where the granted period of each job j_i is precisely τ_i.*

Proof: Feasibility follows by Lemma 4.2. The proof of correctness is identical to the one presented in Corollary 4.6, only here we have $w = t^*$. ∎

Using this version of bal in s&b gives an uslotted version of Algorithm s&b for which we have the following result.

Theorem 4.9 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of the periodic scheduling problem, and assume that there exists a real number $c > 0$ and some nonnegative integers $e, e_1, \ldots, e_n$ such that $\tau_i = c \cdot 2^{e_i}$ for all i. If $t^* = t/2^e$, then the unslotted Algorithm* s&b *with parameter t^* finds a periodic schedule S for J such that $\frac{\tau_i^S}{\tau_i} = f$ for all $i = 1, \ldots, n$ where $f = \beta + R\frac{t}{t^*}$.*

Proof: The proof is identical to the one of Theorem 4.7 when referring to Lemma 4.8 instead of Lemma 4.1. ∎

4.4 A $\frac{9}{8} + O(R)$ Approximation Algorithm for AVE

In this section we present an algorithm for the AVE measure, without any preconditions. This algorithm improves on the known upper bound for the unit-length model presented in [10], and it demonstrates the applicability of Algorithm s&b.

The algorithm is a straightforward application of Algorithm s&b and its analysis relies on a powerful lemma that bounds the approximation factor for AVE in terms of the free bandwidth and bounds on the individual ratios. We start by stating the lemma, which is a variant of the lemma first proven in Lemma 2.5 in [10] for the unit length case.

Lemma 4.10 (Leftover Lemma for unit-size jobs from [10]) *Let $\{a_1, \ldots, a_n\}$ and $\{f_1, \ldots, f_n\}$ be such that for all i, $a_i, f_i > 0$ and $\sum a_i = 1, \sum f_i \leq 1$. Let $\Delta = 1 - \sum f_i$. If for all i, $\rho_l \leq \frac{a_i}{f_i} \leq \rho_h$, for some $\rho_l \leq \rho_h$, then $\sum \frac{a_i^2}{f_i} \leq \rho_l + \rho_h - \rho_l\rho_h + \rho_l\rho_h\Delta$.*

We extend this lemma to the variable size case and change the notation to the one we use.

Lemma 4.11 (Leftover Lemma for variable-size jobs) *Let $\{b_1, \ldots, b_n\}$ be such that for all i, $b_i > 0$. Let $\{\tau_1, \ldots, \tau_n\}, \{\tau'_1, \ldots, \tau'_n\}$ be such that for all i, $\tau_i, \tau'_i > 0$. Let $\sum \frac{b_i}{\tau_i} = 1, \sum \frac{b_i}{\tau'_i} \leq 1$. Let $\Delta = 1 - \sum \frac{b_i}{\tau'_i}$. If for all i, $\rho_l \leq \frac{\tau'_i}{\tau_i} \leq \rho_h$, for some $\rho_l \leq \rho_h$, then $\sum \frac{b_i\tau'_i}{{\tau_i}^2} \leq \rho_l + \rho_h - \rho_l\rho_h + \rho_l\rho_h\Delta$.*

Proof: For all i, define a_i to be $a_i = \frac{b_i}{\tau_i}$ and f_i to be $f_i = \frac{b_i}{\tau'_i}$. Our new sets of $\{a_i\}$ and $\{f_i\}$ with ρ_l, ρ_h agree with the terms of Lemma 4.10 and therefore $\sum \frac{a_i^2}{f_i} = \sum \frac{b_i\tau'_i}{{\tau_i}^2} \leq \rho_l + \rho_h - \rho_l\rho_h + \rho_l\rho_h\Delta$. ∎

Note that if the τ_is are the requested periods and τ'_i are the granted periods, this gives a bound on the approximation factor for the AVE measure.

The Leftover Lemma enables us to analyze the approximation factor of the simple algorithm presented in Figure 4.4 which we call Algorithm A. This is our first result for *any* instance. In Algorithm A we round the periods to the next powers of 2 and then run Algorithm s&b. Rounding the periods to the next powers of 2 is a fundamental technique in the unit-size model and is quite commonly used.

Theorem 4.12 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance for the periodic scheduling problem with requested bandwidth 1, and let S be the schedule produced for J by Algorithm A. Then $C_{\text{AVE}}(J, S) \leq \frac{9}{8} + \frac{3}{2}R + \frac{1}{2}R^2$.*

Proof: Let β' and β_f denote the total bandwidths after steps 1 and 2, respectively. Denote $\min\{\tau'_i\}$ by t' and let $R' = B/t'$, i.e., R' is the extent of the instance submitted to s&b in step 2. Let t_i denote the granted period of job j_i. Clearly, $1 \leq \frac{\tau'_i}{\tau_i} < 2$. Step

Algorithm A

Input: Instance J.

Output: Schedule S.

Code:

(1) Round the requested periods τ_i up to the nearest powers of 2: let $\tau_i' = 2^{\lceil \log \tau_i \rceil}$.

(2) Apply Algorithm s&b to the jobs with requested periods τ_i' and parameter $t^* = \min\{\tau_i'\}$.

Figure 4.4: Algorithm A.

2 just multiplies the τ_i's by $f_0 = \lfloor ft' \rfloor / t'$ where $f = \beta' + R'$ and hence $\frac{t_i}{\tau_i'} = f_0$ and $\rho_l \leq \frac{t_i}{\tau_i} \leq \rho_h$ for $\rho_l = f_0$ and $\rho_h = 2f_0$. Also note that $\beta_f = \beta'/f_0$. From the Leftover Lemma (Lemma 4.11) we get

$$\begin{aligned} C_{\mathrm{AVE}}(J,S) &\leq f_0 + 2f_0 - 2f_0^2 + 2f_0^2\Delta_f \\ &= 3f_0 - 2f_0^2\beta_f \\ &= f_0(3 - 2\beta') \\ &\leq f(3 - 2\beta') \\ &= (\beta' + R')(3 - 2\beta') \\ &= -2\beta'^2 + (3 - 2R')\beta' + 3R' \ . \end{aligned}$$

Using elementary calculus we find that the latter expression is maximized when $\beta' = \frac{3-2R'}{4}$, and since $R' \leq R$, we get that the worst-case performance is therefore

$$\begin{aligned} C_{\mathrm{AVE}}(J,S) &\leq (\beta' + R')(3 - 2\beta') \\ &\leq \frac{1}{8}(3 + 2R')^2 \\ &\leq \frac{1}{8}(3 + 2R)^2 \\ &= \frac{9}{8} + \frac{3}{2}R + \frac{1}{2}R^2 \ . \quad \blacksquare \end{aligned}$$

Note that it follows from the analysis that Algorithm A gives a $2(1+R)$ approximation for MAX by using ρ_h as a bound on the approximation factor since $f_0 \leq 1 + R$.

An unslotted version of this algorithm is trivially obtained by using the unslotted version of Algorithm s&b.

An algorithm that achieves an asymptotic approximation ratio of $\frac{9}{8}$ for the unit-size case is presented in [10] (referred to as Algorithm B). Comparing the two algorithms shows the following.

- Our Algorithm A is more general since it does not require that all jobs have the same length.
- "Algorithm B" works only when $a_1 < \frac{1}{42}$ where a_1 is the maximal share request. Our Algorithm A has no such restriction.
- The approximation ratio Algorithm A achieves is better than the one of "Algorithm B". The approximation ratio for the AVE measure of "Algorithm B" is $\frac{9}{8-20a_1} = \frac{9}{8}\sum_{i=0}^{\infty}\left(\frac{5}{2}a_1\right)^i > \frac{9}{8} + \frac{45}{16}a_1 + \frac{225}{32}a_1^2$ which is greater than $\frac{9}{8} + \frac{3}{2}R + \frac{1}{2}R^2$ (in the unit-size model, R is the maximal share request).
- We believe that the description and analysis of Algorithm A are simpler than the ones of "Algorithm B".

Chapter 5

A $1 + \frac{\sqrt{2}}{2} + O(R)$ Approximation Algorithm for MAX

In previous chapters we created "building blocks" for algorithms and showed how we can get asymptotic approximation of 2 for the MAX measure using nothing but simple rounding. In this chapter, we show that using a little more sophisticated rounding, we can achieve an asymptotic approximation ratio of $1 + \frac{\sqrt{2}}{2} < 2$.

Previous work gave very little attention to the MAX measure. No algorithm with asymptotic approximation of less than 2 has been presented so far for the MAX measure, even for the case of unit-size jobs.

The algorithm presented in this chapter, which we call Algorithm B, is based on Algorithm s&b, presented in Chapter 4, and shows the power of Algorithm s&b for the MAX measure. The algorithm is presented in Figure 5.1.

The algorithm tries two forms of rounding. First we use the simple rounding of Algorithm A and round all the periods to the next power of 2. Then we try a different form of rounding, we round $\frac{\tau_i}{\sqrt{2}}$ to the next power of 2 and try to run Algorithm s&b on the resulting output. Our analysis shows that the best result of the two gives an approximation of at most $1 + \frac{\sqrt{2}}{2} + 2R$.

Theorem 5.1 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^{n}$ be an instance for the periodic scheduling problem with requested bandwidth 1, and let S be the schedule produced for J by Algorithm* B. *Then $C_{MAX}(J, S) \leq 1 + \frac{\sqrt{2}}{2} + 2R$.*

Intuitively, it appears that step 2 can do better than step 1 only when the rounded jobs $\{(b_i : \tau_i')\}$ of step 1 have a "small" unused bandwidth. In such a case, we are guaranteed that not many of the jobs (in terms of bandwidth) were rounded up and therefore the cost of rounding *down* some of the jobs is not too big. In this case we round some of the

Algorithm B

Input: Instance J.

Output: Schedule S.

Code:

(1) Let $\tau_i' = 2^{\lceil \log \tau_i \rceil}, t' = \min\{\tau_i'\}$. Run Algorithm s&b on $\{(b_i : \tau_i')\}$ with parameter t' and denote the result by S_1.

(2) Let $\tau_i'' = 2^{\lceil \log(\tau_i) - \frac{1}{2} \rceil}, t'' = \min\{\tau_i''\}$. Run Algorithm s&b on $\{(b_i : \tau_i'')\}$ with parameter t'' and denote the result by S_2.

(3) If $C_{MAX}(J, S_1) \leq C_{MAX}(J, S_2)$, return S_1. Otherwise, return S_2.

Figure 5.1: Algorithm B.

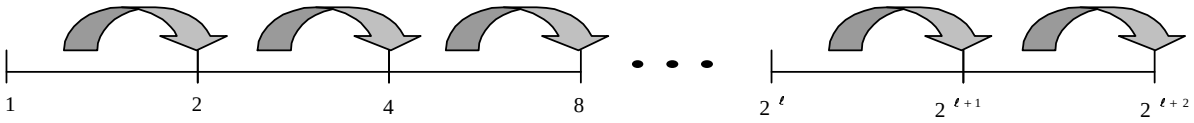


Figure 5.2: First Step: Round up all jobs.

jobs back down, which is equivalent (as shown below) to rounding τ_i to $\tau_i'' = 2^{\lceil \log(\tau_i) - \frac{1}{2} \rceil}$.

Proof: In the analysis we bound the performance of the algorithm by distinguishing between two cases. We show that in the first case, running step 1 gives the required approximation ratio and in the second, running step 2 gives the required approximation ratio. Therefore taking the best of the two gives the required approximation ratio for all cases.

We analyze the algorithm using a parameter δ that is later optimized to achieve the best approximation ratio possible.

Consider the rounding performed in step 1 of the algorithm. The periods are rounded to the next powers of 2 (see Figure 5.2). Denote the bandwidth of the new instance by β', its extent by R' and its free bandwidth by $\Delta' = 1 - \beta'$. The two cases of the analysis are for $\Delta' \geq \delta$ and for $\Delta' < \delta$. For the first case, $\Delta' \geq \delta$, we assume that we have a "relatively large" Δ' and therefore using Algorithm s&b on the rounded instance should give a small enough approximation factor. If $\Delta' < \delta$, we work more carefully. Since we have a "relatively small" Δ', the bandwidth of the elements that were rounded up "by much" is quite small. Therefore we can take these elements and round them down instead, while not increasing the total bandwidth by too much. This way we get an instance where no element has been rounded up "by much" and improve the approximation factor for the MAX measure.

The First Case (Large Δ'): Suppose $\Delta' \geq \delta$. In this case, we apply Algorithm s&b on the rounded instance $\{(b_i : \tau_i')\}$. By Theorem 4.7, the algorithm gives an approximation

ratio of at most $1 - \Delta' + R' \leq 1 - \delta + R$. Since the rounding performed was by a factor of at most 2, we have an overall approximation factor of $C_{MAX}(J, S_1) \leq 2 - 2\delta + 2R$.

The Second Case (Small Δ'): Suppose $\Delta' < \delta$. For some parameter $\gamma \in [1, 2]$, we divide the set of jobs into two subsets, G_1 and G_2, such that

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

We denote the bandwidth of G_1, G_2 by β_1, β_2 respectively. Note that rounding to τ'_i increases the requested period by a factor of $2^{1-(\log \tau_i - \lfloor \log \tau_i \rfloor)}$. Therefore, G_1 is the set of jobs with "big rounding" in step 1 and G_2 is the set of jobs with "small rounding" in step 1. We claim that β_1 is "relatively small" since if G_1 had significant bandwidth, we would get a large Δ'. The following lemmas formalize and prove this claim.

Lemma 5.2 $\beta' \leq \frac{\gamma}{2}\beta_1 + \beta_2$.

Proof: Let β'_1, β'_2 denote the bandwidth of G_1, G_2 respectively after performing the rounding of step 1 of the algorithm. An element of G_1 must be rounded by a factor of at least $\frac{2}{\gamma}$ and therefore $\beta'_1 \leq \frac{\gamma}{2}\beta_1$. For elements of G_2, it is possible that no rounding is performed at all and therefore $\beta'_2 \leq \beta_2$ holds and is tight. ∎

Lemma 5.3 $\beta_1 \leq \frac{2}{2-\gamma}\delta$.

Proof: By Lemma 5.2 we have $\beta' \leq \frac{\gamma}{2}\beta_1 + \beta_2$. Since $\Delta' \leq \delta$ we have $\beta' \geq 1 - \delta$. We get $\frac{\gamma}{2}\beta_1 + \beta_2 \geq 1 - \delta$ and therefore $\delta \geq (1 - \beta_2) - \frac{\gamma}{2}\beta_1 = \left(1 - \frac{\gamma}{2}\right)\beta_1$. Simple algebra shows that $\beta_1 \leq \frac{2}{2-\gamma}\delta$. ∎

We now round G_1 *downwards* (see Figure 5.3). By Lemma 5.3, this will not increase the total bandwidth dramatically. The following lemma formalizes this claim. Denote the periods after rounding by τ^γ. Formally we define

$$\tau_i^\gamma = \begin{cases} 2^{\lceil \log \tau_i \rceil - 1} & \text{if } i \in G_1, \\ 2^{\lceil \log \tau_i \rceil} & \text{if } i \in G_2. \end{cases}$$

Denote the bandwidth of the instance $\{(b_i : \tau_i^\gamma)\}$ by β^γ, denote its extent by R^γ and denote the bandwidths of G_1, G_2 when using the new rounding by $\beta_1^\gamma, \beta_2^\gamma$. Note that rounding G_1 downwards and G_2 upwards is equivalent to rounding $\frac{\tau_i}{\gamma}$ upwards since if $i \in G_1$, dividing it by γ and then rounding will get the same result as rounding down. Therefore, τ^γ for $\gamma = \sqrt{2}$ is equivalent to τ'' in step 2 of the algorithm.

Lemma 5.4 $\beta^\gamma \leq \gamma\beta_1 + \beta_2$.

Proof: The jobs in G_1 are rounded down by a factor of at most γ and therefore $\beta_1^\gamma \leq \gamma\beta_1$. The jobs in G_2 are still rounded up and still it is possible that the requested

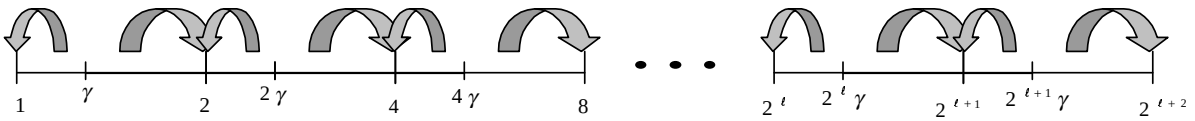


Figure 5.3: The Second Case (Small Δ'): Round down the jobs of G_1 and round up the jobs of G_2.

period of a job in G_2 remains the same. Therefore $\beta_2^\gamma \leq \beta_2$ holds and is tight. We get $\beta^\gamma = \beta_1^\gamma + \beta_2^\gamma \leq \gamma\beta_1 + \beta_2$. ∎

When we apply Algorithm s&b to the instance obtained, we get, by Theorem 4.7, an approximation factor of at most

$$\begin{aligned}\beta^\gamma + R^\gamma &\leq \gamma\beta_1 + \beta_2 + R^\gamma \\ &= \gamma\beta_1 + (1-\beta_1) + R^\gamma \\ &= (\gamma-1)\beta_1 + 1 + R^\gamma \ .\end{aligned}$$

We use Lemma 5.3 to bound β_1. Rounding G_1 downwards can only increase R by a factor of γ and therefore $\beta^\gamma + R^\gamma \leq 2\delta\frac{\gamma-1}{2-\gamma} + 1 + \gamma R$. The approximation factor of the algorithm is at most the maximal rounding up performed, times $\beta^\gamma + R^\gamma$. The maximal rounding up is no more than $\frac{2}{\gamma}$ since only G_2 is rounded up and a job of G_2 can only be rounded by a factor of at most $\frac{2}{\gamma}$. We get

$$\begin{aligned}C_{MAX}(J,S_2) &\leq \frac{2}{\gamma}\left(\beta^\gamma + R^\gamma\right) \\ &\leq \frac{2}{\gamma}\left(\frac{2(\gamma-1)}{2-\gamma}\delta + 1 + \gamma R\right) \\ &= \frac{2}{\gamma}\left(2\delta\frac{\gamma-1}{2-\gamma} + 1\right) + 2R \ .\end{aligned}$$

We choose the value of γ that minimizes this expression. Denote $g(\gamma) = \frac{4(\gamma-1)}{\gamma(2-\gamma)}\delta + \frac{2}{\gamma} + 2R$. The minimal value of g is found using the derivative. We get a minimal value of g when using $\gamma = 2\sqrt{1-\delta}\frac{\sqrt{1-\delta}-\sqrt{\delta}}{1-2\delta}$. This minimal value is $g\left(2\sqrt{1-\delta}\frac{\sqrt{1-\delta}-\sqrt{\delta}}{1-2\delta}\right) = 2\frac{\sqrt{1-\delta}(1-2\delta)}{\sqrt{1-\delta}-\sqrt{\delta}} - (1-2\delta) + 2R$ and the approximation factor of the algorithm in this case is $C_{MAX}(J,S_2) \leq 2\frac{\sqrt{1-\delta}(1-2\delta)}{\sqrt{1-\delta}-\sqrt{\delta}} - (1-2\delta) + 2R$.

After analyzing the two cases for any value of δ, we pick a value of δ that minimizes the overall approximation ratio for both cases. For the first case we have shown an approximation factor of $C_{MAX}(J,S_1) \leq 2 - 2\delta + 2R$ and for the second case we have shown an approximation factor of $C_{MAX}(J,S_2) \leq 2\frac{\sqrt{1-\delta}(1-2\delta)}{\sqrt{1-\delta}-\sqrt{\delta}} - (1-2\delta) + 2R$. Therefore, for a general input, the worst-case approximation ratio is $C_{MAX}(J,S) \leq \max\left\{2 - 2\delta + 2R, 2\frac{\sqrt{1-\delta}(1-2\delta)}{\sqrt{1-\delta}-\sqrt{\delta}} - (1-2\delta) + 2R\right\}$. The value of δ that minimizes this

expression is the one that makes the terms for the two cases equal, that is when $2 - 2\delta + 2R = 2\frac{\sqrt{1-\delta}(1-2\delta)}{\sqrt{1-\delta}-\sqrt{\delta}} - (1 - 2\delta) + 2R$. We get $\delta = \frac{1}{2} - \frac{\sqrt{2}}{4}$. This value of δ gives $\gamma = \sqrt{2}$ and therefore $\tau'' = \tau^{\gamma=\sqrt{2}}$ gives the required performance in the second case.

We get a worst-case approximation ratio of $C_{MAX}(J, S) \leq 1 + \frac{\sqrt{2}}{2} + 2R$. ∎

Note that the description and analysis of Algorithm B apply for both the slotted and unslotted models depending on the version of Algorithm s&b used.

Chapter 6

Separable Schedules

In this chapter we introduce an additional new technique for periodic schedules, based on the concept of *Separable Schedules*. The notion of separable schedules is new and first presented in this work. Separable schedules are an abstraction of the schedules produced by the s&b algorithm. We show how to perform certain operations on them while bounding the approximation factor. We use these operations in Chapter 7 to create better approximation algorithms.

6.1 Definition of Separable Schedules

Definition 6.1 *A schedule S is called* separable *if S can be partitioned into equal-size runs of time slots called* bins *such that the following conditions hold true.*

(1) Each occurrence of a job that starts in some bin z ends in bin z.

(2) Each job appears at most once in each bin.

(3) If a job j starts in some bin z, there are no idle time slots before j in z. Furthermore, all jobs that start in z before j, occur in each bin j occurs in, and they all start before j in each of these bins.

(4) The occurrence of jobs in bins is periodic. That is, 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 4 and 3.

Lemma 6.1 *A separable schedule is periodic.*

Proof: From property 4 it follows that what needs to be shown is that in all bins a job j appears in, it has the same offset. Since by property 3, the same sequence of

Algorithm P

Input: Separable schedule S with bin size w, natural parameter p.

Output: Schedule S'.

Code:

(1) Partition each bin into p parts of size w/p each.

(2) Scan the bin parts in order. For each bin part z in turn:

2a: If the last job in z is not completely contained in z, add it to z and remove it from $z+1$.

2b: The schedule associated with z is all jobs in order (aligned to the beginning of z) followed by idle time slots as to get total length of $w/p + B$ slots.

Figure 6.1: Algorithm P.

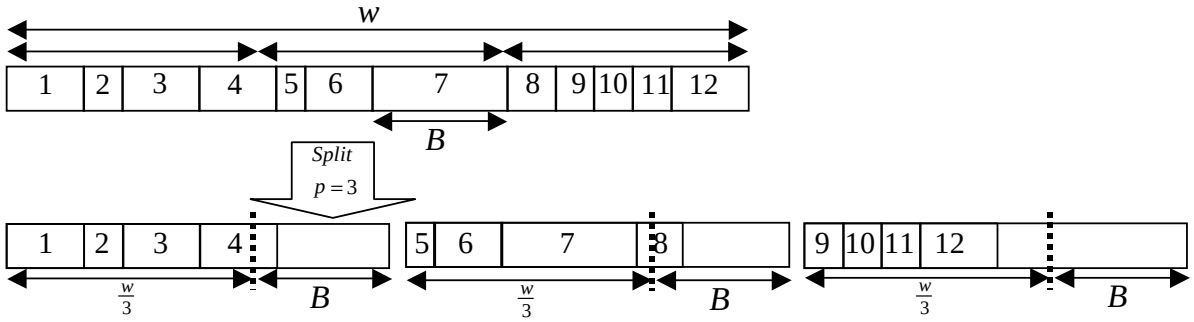


Figure 6.2: Example of Algorithm P with parameter $p = 3$.

jobs appears in all bins before j with no spaces, it has the same offset in all bins and periodicity follows. ∎

6.2 Operations on Separable Schedules

We now present Algorithm P (see Figure 6.1) which splits a separable schedule, creating another separable schedule with smaller bins and larger periods. The algorithm is given, as input, a separable schedule S with bins of size w, and a natural number p.

An example of how the algorithm runs appears in Figure 6.2. The following lemma summarizes the properties of Algorithm P.

Lemma 6.2 *Let S be a separable schedule with bin size w. Then for any given p, Algorithm* P *outputs a separable schedule S' with bin size $\frac{w}{p} + B$ such that $\tau_i^{S'} = (1 + \frac{pB}{w})\tau_i^S$.*

Proof: *Separability:* From our construction it is clear that since S is separable, no job

can appear in a bin of S' more than once and a job finishes at the same bin it started in and therefore properties 1 and 2 of Definition 6.1 hold.

For any job i, if i was assigned to the kth bin-part of its bin in S, it will be assigned to the kth bin-part in all bins of S it appears in. This is since the prior elements in the bins of S are the same by property 3 of S and since the association of jobs with bin-parts is performed in-order. This argument is true for all of the predecessors of i as well and since the predecessors of i are the same in all bins of S it appears in, it will also have the same predecessors in all bins of S' it appears in. Therefore property 3 holds for S'. Combined with property 4 of S, this claim also gives property 4 for S': since job 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 i appears are also periodic.

Period: Assume a job has an original period $\tau_i^S = k \cdot w$. Its new period in S' is $\tau_i^{S'} = k \cdot p \cdot \left(\frac{w}{p} + B\right) = \left(1 + \frac{pB}{w}\right) \tau_i^S$ since each bin of size w was split into p bins of size $\frac{w}{p} + B$. ∎

Note that Algorithm P as described here works at an unslotted model. We prefer to use an unslotted model here in order to simplify the mathematics of the algorithms below. We create a slotted version of Algorithm P by truncating the bin size of the new schedule to $\left\lfloor \frac{w}{p} + B \right\rfloor$. In such case we get $\tau_i^{S'} = \frac{p}{w} \left\lfloor \frac{w}{p} + B \right\rfloor \tau_i^S \leq (1 + \frac{pB}{w}) \tau_i^S$. Periodicity is preserved by the lemma below.

Lemma 6.3 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of periodic scheduling where b_i, τ_i are integral 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' which is periodic and for all i, $\tau_i^{S'} = \frac{\lfloor w \rfloor}{w} \tau_i^S \leq \tau_i^S$.*

Proof: From the integrality of the input and from property 3 in Definition 6.1, it follows that truncating the bins gives a feasible schedule of J. This is because in each bin of S we have jobs of integral length followed by (possibly) idle time and thus truncating the bin to the nearest integral size does not cause any job to end outside the bin. The new schedule S' complies with Definition 6.1 since the bins of S' contain the same jobs as the bins of S and in the same order.

For any j_i, let u_i denote τ_i^S / w. From properties 2 and 3 of Definition 6.1 it follows that u_i is integral. In S', the period of j_i is $\tau_i^{S'} = u_i \cdot \lfloor w \rfloor$ since the length of a bin is now $\lfloor w \rfloor$ and the offset within the bin remains unchanged. Therefore $\frac{\tau_i^{S'}}{\tau_i^S} = \frac{\lfloor w \rfloor}{w}$. ∎

Next, we define Algorithm M (see Figure 6.3) that performs the operation of merging separable schedules. This time, the resulting schedule is not necessarily separable. The input is k separable schedules $S_1, \ldots, S_k$ with bin sizes $w_1, \ldots, w_k$, respectively. The job sets of the schedules are disjoint. An example of how the algorithm runs can be found

Algorithm M

Input: Separable schedules $S_1, \ldots, S_k$.

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, followed by the second bin in S_1 etc.

Figure 6.3: Algorithm M.

$$
\begin{aligned}
S_0 &: \; Y_{0,1}, Y_{0,2}, Y_{0,3}, \ldots, Y_{0,\ell_0} \\
S_1 &: \; Y_{1,1}, Y_{1,2}, Y_{1,3}, \ldots, Y_{1,\ell_1} \\
&\vdots \\
S_{k-1} &: \; Y_{k-1,1}, Y_{k-1,2}, Y_{k-1,3}, \ldots, Y_{k-1,\ell_{k-1}} \\
\hline
S_{merged} &: \; Y_{0,1}, Y_{1,1}, \ldots, Y_{k-1,1}, Y_{0,2}, Y_{1,2}, \ldots, Y_{k-1,2}, \ldots
\end{aligned}
$$

Figure 6.4: Example of Algorithm M. The lth bin in the ith schedule is denoted by $Y_{i,l}$.

in Figure 6.4. We have the following result.

Lemma 6.4 *Let $S_1, \ldots, S_k$ be separable schedules with bin sizes $w_1, \ldots, w_k$, respectively. Then Algorithm* M *outputs a periodic schedule S such that for all $j \in S_i$ we have $\tau_j^S = \frac{W}{w_i}\tau_j^{S_i}$, where $W = \sum_{i=1}^{k} w_i$.*

Proof: Let j be some job of S_i. The period of j in S_i is $\tau_j^{S_i} = k \cdot w_i$. This means j appears every kth bin of S_i, with the same offset in every bin. In the merged schedule S, a bin of j_i appears every W time slots and the bins appear in order. Therefore, a bin that j appears in is scheduled every $k \cdot W$ time slots in S. Therefore S is periodic for j and the period of j in S is $k \cdot W$. We get $\tau_j^S = \frac{W}{w_i} \cdot \tau_j^{S_i}$. ∎

Finally, we prove that the schedules produced by Algorithm s&b are separable. This property is later used to apply the spilt and merge operations on schedules produced by Algorithm s&b. Note that we prove the claim for the unslotted model, while the extension to the slotted case is trivial.

Lemma 6.5 *Let J be an instance of the periodic scheduling problem. Let $t^* = t_J/2^e$ for some nonnegative integer e. Then Algorithm* s&b *with parameter t^* produces a separable schedule with bin size $t^*(\beta_J + R_J\frac{t_J}{t^*}) = t^*\beta_J + B_J$.*

Proof: Consider the schedule generated by s&b, and consider each leaf as a bin. By construction, each bin has size $t^*(\beta_J + R_J\frac{t_J}{t^*}) = t^*\beta_J + B_J$, and hence property 1 of

Definition 6.1 holds. Property 2 follows from the correctness of step 3b in Algorithm bal: The sum of lengths of the jobs in each leaf is at most $t^*(\beta_J + R_J \frac{t_J}{t^*})$ and therefore any job that starts in some leaf ends in that leaf. Property 3 holds by Lemma 4.3. Property 4 follows from Lemma 4.5 and the fact that the granted periods of all jobs are divided by the bin size. ∎

Chapter 7

A General Algorithm for MAX

In this chapter we present our general algorithm for the MAX measure. The algorithm is presented using parameters k and L that are determined later. Since the analysis is a bit complicated, we first present the single server case in Section 7.1, and then explain how to generalize it to multiple servers in Section 7.2.

7.1 The Single Server Case

In Figure 7.1 we present an algorithm, which we call Algorithm C, that works for the unslotted model, we then describe how to adapt the algorithm to the slotted case in subsection 7.1.1.

Theorem 7.1 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of periodic scheduling, and let S be the schedule produced by Algorithm C for J with parameters k, L. Then $C_{\text{MAX}}(J, S) \leq (1 + \frac{1}{k}) \cdot (1 + \frac{1}{L}) \cdot (1 + 2k(L+1)R_J)$.*

Proof: First, observe 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 up to a common factor of $2^{\frac{\ell}{k}}$. In addition, the minimal period of jobs in G_ℓ is the minimal period in G_{ℓ^*} times $2^{e+\frac{\ell-\ell^*}{k}}$ for some integer e, and hence t_ℓ is a power of $\frac{1}{2}$ times the minimal period in G_ℓ.

Next, we analyze the approximation factor. Step 1 contributes a factor of at most $2^{\frac{1}{k}}$. For Step 4, let β_ℓ denote the total bandwidth of jobs in G_ℓ. By Theorem 4.9, Step 4 increases the periods of jobs in G_ℓ by a factor $f_\ell = \beta_\ell + \frac{B}{t_\ell}$. Define $r_\ell = B/t_\ell$. Note that r_ℓ is an upper bound on the extent of the jobs in G_ℓ. Let $R_\ell = r_\ell/f_\ell$: R_ℓ is an upper bound on the extent of the jobs in S_ℓ. To analyze Step 5, note that by Lemma 6.2 and Lemma 6.5, the periods are increased by a factor of $1 + \frac{p_\ell B}{t_\ell f_\ell} = 1 + p_\ell R_\ell$. The

Algorithm C

Input: Instance J, parameters k, L.

Output: Schedule S.

Code:

(1) Round the requested periods up to the next powers of $2^{\frac{1}{k}}$. Formally, let $\tau'_i = 2^{\frac{1}{k}\lceil k \log(\tau_i)\rceil}$.

(2) Partition the jobs into k classes $G_0, \ldots, G_{k-1}$ according to their τ' values: Job j is in G_ℓ if $\tau'_j = 2^{e+\frac{\ell}{k}}$ for some integer e.

(3) Let $t' = \min\left\{\tau'_j \mid j \in J\right\}$. Let ℓ^* be such that $G_{\ell^*} \ni j$ for some job j with $\tau'_j = t'$. Define $t_0 = \frac{t'}{2^{\ell^*/k}}$, and for $\ell = 1, \ldots, k-1$, define $t_\ell = t_0 \cdot 2^{\ell/k}$.

(4) Apply Algorithm s&b to each class G_ℓ with parameter t_ℓ. Let S_ℓ denote the resulting schedule for class G_ℓ.

(5) Apply Algorithm P to each schedule S_ℓ, with parameter $p_\ell = \left\lceil L \cdot 2^{\ell/k} \right\rceil$.

(6) Apply Algorithm M to the k schedules produced, and output the resulting schedule.

Figure 7.1: Algorithm C.

latter expression can be bounded as follows.

$$\begin{aligned} 1 + p_\ell R_\ell &\le 1 + (L2^{\ell/k} + 1)\frac{r_\ell}{f_\ell} \\ &= \frac{\beta_\ell + 2r_\ell + Lr_0}{f_\ell}. \end{aligned}$$

To analyze Step 6, we compute the bin size w_ℓ for each S_ℓ produced by Step 5. On the one hand, since $\left\lceil L2^{\ell/k} \right\rceil \ge L2^{\ell/k}$, we get

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

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

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

Hence we have that the sum of the bin sizes W is at most

$$\begin{aligned} W &= \sum_{\ell=0}^{k-1} w_\ell \\ &\leq \sum_{\ell=0}^{k-1} \frac{t_0}{L}(\beta_\ell + r_\ell + Lr_0) \\ &\leq \frac{t_0(1+kr_0(1+L))}{L} \ . \end{aligned}$$

Now we can apply Lemma 6.4 to get that Step 6 increases the periods by at most

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

To conclude, we multiply together all factors affecting the periods, and find that

$$\begin{aligned} C_{\text{MAX}}(J,S) &\leq 2^{1/k} \cdot f_\ell \cdot (1+p_\ell R_\ell)\frac{W}{w_\ell} \\ &\leq 2^{1/k}(1+2^{-\ell/k}L^{-1})(1+k(r_0+Lr_0)) \\ &= 2^{1/k}(1+2^{-\ell/k}L^{-1})(1+k(1+L)r_0) \\ &\leq \frac{L+1}{L} \cdot \frac{k+1}{k} \cdot (1+2kR_J(1+L)) \ . \end{aligned}$$

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

Corollary 7.2 *For any instance J of the periodic scheduling problem, there exists a schedule S such that*

$$C_{\text{MAX}}(J,S) \leq 1 + 3(2R_J)^{\frac{1}{3}} + O(R_J^{\frac{2}{3}}) < 1 + 3.78{R_J}^{\frac{1}{3}} + O(R_J^{\frac{2}{3}}) \ .$$

Proof: Apply Algorithm C with parameters $k = L = (2R_J)^{-\frac{1}{3}}$. ∎

7.1.1 A Slotted Version of Algorithm C

We now explain how to adapt Algorithm C to the slotted case. In the slotted case, each job has integral length and a job must start at some integral time slot. A slotted version of Algorithm C runs as follows. It runs Algorithm C but uses the slotted version of Algorithm P in step 5 of the algorithm. 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 C and the feasibility of the slotted Algorithm P.

We now show that this algorithm produces results that are at least as good as the ones of the unslotted Algorithm C.

Theorem 7.3 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of periodic scheduling where b_i, τ_i are integral for all i, and let S be the schedule produced by the slotted Algorithm* C *for J with parameters k, L. Then S is slotted and $C_{\text{MAX}}(J, S) \leq (1 + \frac{1}{k}) \cdot (1 + \frac{1}{L}) \cdot (1 + 2k(L+1)R_J)$.*

Proof: 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 points. Since all job lengths are integral and the jobs of a bin are scheduled with no spaces, all jobs start at integral time slots. Therefore S is a slotted schedule.

Denote the schedules produced in steps 5 and 6 of the unslotted Algorithm C by $S_0^*, \ldots, S_{k-1}^*, S^*$. Consider a job $j_i \in G_\ell$. Up to step 5 of the algorithm, the slotted and unslotted versions are equivalent. In step 5, j_i is scheduled with a period of $\tau_i^{S_\ell^*} = u_i \cdot w_\ell$ in the unslotted version and $\tau_i^{S_\ell} = u_i \cdot \lfloor w_\ell \rfloor$ in the slotted version. By Lemma 6.4, after step 6, j_i has a period of $\tau_i^{S^*} = u_i \cdot W$ in the unslotted case and $\tau_i^S = u_i \cdot \sum_{\ell=0}^{k-1} \lfloor w_\ell \rfloor$ in the slotted one and therefore $\tau_i^S \leq \tau_i^{S^*}$. Therefore the approximation factor of the slotted version is at least as good as the one of the unslotted version and the result follows from Theorem 7.1. ∎

7.2 The Multiple Servers Case

It is almost straightforward to generalize Algorithm C above to the case of m servers. The differences are in steps 5 and 6. In step 5, the new algorithm splits the schedules using parameters $p_\ell = m \left\lceil L 2^{\frac{\ell}{k}} \right\rceil$. The result is that for each class G_ℓ, we get a number of bins which is a multiple of m. Now we can take each "block" of m consecutive bins and multiplex it among our m machines. 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 P. The new algorithm, Algorithm genP, takes two arguments m, p such that $m|p$ and an input separable schedule S and creates m separable schedules $S_0, \ldots, S_{m-1}$. The algorithm is presented in Figure 7.2.

The following lemma summarizes the properties of Algorithm genP.

Lemma 7.4 *Let S be a separable schedule with bin size w. Then for any given m, p such that $m|p$, Algorithm* genP *outputs separable schedules $S_0, \ldots, S_{m-1}$ with bin size $\frac{w}{p} + B$ such that $\tau_i^{S_\ell} = \frac{1}{m}(1 + \frac{pB}{w})\tau_i^S$ for all $j_i \in S_\ell$.*

Algorithm genP

Input: Schedule S, parameters m, p s.t. $m|p$.

Output: Schedules $S_0, \ldots, S_{m-1}$.

Code:

(1) Apply Algorithm P with parameter p on S and enumerate the resulting bins in order.

(2) For each ℓ, the output schedule S_ℓ is a concatenation of all bins whose index is congruent to ℓ modulo m.

Figure 7.2: Algorithm genP.

Proof: Let S' denote the schedule created in Step 1 of the algorithm. By Lemma 6.2, this is a separable schedule with bin size $\frac{w}{p} + B$ and $\tau_i^{S'} = (1 + \frac{pB}{w})\tau_i^S$. Therefore for all jobs j_i we are guaranteed that its offset inside the 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 2 of Algorithm P. Assume bin z has originated from bin z^* of S and bin y originated from bin y^* of S (obviously, $z^* \neq y^*$ 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 2 of Algorithm P assigns a bin to j_i based only on its predecessors (which are identical in z^* and y^*), it follows that for some $u < m$: $z = p(z^* - 1) + u$ and $y = p(y^* - 1) + u$. Therefore $z - y = p(z^* - y^*)$ and $m|(z-y)$ since $m|p$. The periods in S_ℓ are the periods in S', reduced by a factor of m as for all m equal sized bins in S', there is only one such bin in S_ℓ. ∎

Now we can specify the generalized algorithm, which we call Algorithm genC. The algorithm is identical to Algorithm C, except for the following modifications. In step 5, we apply Algorithm genP to each class G_ℓ with parameters m and $p_\ell = m \left\lceil L2^{\frac{\ell}{k}} \right\rceil$, thus obtaining m schedules for each class. Let S_{ℓ_i} denote the ith schedule of class G_ℓ. Then, in step 6, the algorithm produces m schedules by m applications of Algorithm M: the ith application merges all schedules S_{ℓ_i}, ranging over all ℓ.

Note that Algorithm genC is a generalization of Algorithm C: for $m = 1$ we get Algorithm C precisely.

The generalization of the approximation factor analysis is very similar to the one shown in Theorem 7.1.

Theorem 7.5 *Let $J = \{j_i = (b_i : \tau_i)\}_{i=1}^n$ be an instance of periodic scheduling for m machines, and let S be the schedule produced by Algorithm* genC *for J with parameters*

k, L. *Then*

$$C_{\text{MAX}}(J,S) \leq \left(1+\frac{1}{k}\right)\cdot\left(1+\frac{1}{L}\right)\cdot\left(1+2k\left(L+\frac{1}{m}\right)R_J\right).$$

Proof: The analysis is very similar to the one of Theorem 7.1. The feasibility argument is identical to the one of Theorem 7.1 above while using Lemma 7.4 instead of Lemma 6.2. As to the approximation factor: as above, Step 1 contributes a factor of at most $2^{\frac{1}{k}}$. For Step 4, let β_ℓ denote the total bandwidth of jobs in G_ℓ as before. By Theorem 4.9, Step 4 increases the periods of jobs in G_ℓ by a factor of $f_\ell = \beta_\ell + \frac{B}{t_\ell}$. Define $r_\ell = B/t_\ell$ and $R_\ell = r_\ell/f_\ell$ as above. To analyze Step 5, note that by Lemma 7.4, the periods are increased by a factor of $\frac{1}{m}\left(1+\frac{p_\ell B}{t_\ell f_\ell}\right) = \frac{1}{m}(1+p_\ell R_\ell)$. The latter expression can be bounded as follows.

$$\begin{aligned}\frac{1}{m}(1+p_\ell R_\ell) &\leq \frac{1}{m}\left(1+m(L2^{\ell/k}+1)\right)\frac{r_\ell}{f_\ell}\\ &= \frac{\beta_\ell+(m+1)r_\ell+mLr_0}{mf_\ell}.\end{aligned}$$

To analyze Step 6, we compute the bin size w_ℓ for each S_ℓ produced by Step 5. On the one hand, since $\left\lceil L2^{\ell/k}\right\rceil \geq L2^{\ell/k}$, we get

$$\begin{aligned}w_\ell &\leq \frac{f_\ell t_\ell}{mL\cdot 2^{\ell/k}} + B\\ &= \frac{t_0}{mL}(\beta_\ell + r_\ell + mLr_0)\ .\end{aligned}$$

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

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

Hence we have that the sum of the bin sizes W is at most

$$\begin{aligned}W &= \sum_{\ell=0}^{k-1} w_\ell\\ &\leq \sum_{\ell=0}^{k-1}\frac{t_0}{mL}(\beta_\ell + r_\ell + mLr_0)\\ &\leq \frac{t_0(m+kr_0(1+mL))}{mL}\ .\end{aligned}$$

Now we can apply Lemma 6.4 to get that Step 6 increases the periods by at most $\frac{W}{w_\ell}$. This size is bounded by:

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

To conclude, we multiply together all factors affecting the periods, and find that

$$\begin{aligned}C_{\text{MAX}}(J, S) &\leq 2^{1/k} \cdot f_\ell \cdot \frac{1}{m}(1 + p_\ell R_\ell)\frac{W}{w_\ell} \\ &\leq 2^{1/k}(1 + 2^{-\ell/k}L^{-1}) \left(1 + k\left(\frac{1}{m} + L\right) r_0\right) \\ &\leq \frac{L + 1}{L} \cdot \frac{k + 1}{k} \cdot \left(1 + 2kR_J\left(\frac{1}{m} + L\right)\right) \ .\end{aligned}$$

Since as before $2^{\frac{1}{k}} \leq 1 + \frac{1}{k}$ for $k \geq 1$, and $r_0 < 2R_J$. ∎

Corollary 7.6 *For any instance J of the periodic scheduling problem on m servers, there exists a schedule S such that*

$$C_{\text{MAX}}(J, S) \leq 1 + 3(2R_J)^{\frac{1}{3}} + O(R_J^{\frac{2}{3}}) < 1 + 3.78{R_J}^{\frac{1}{3}} + O(R_J^{\frac{2}{3}}) \ .$$

Proof: Apply Algorithm genC with parameters $k = L = (2R_J)^{-\frac{1}{3}}$. ∎

Chapter 8

Conclusions and Discussion

In this work a more general model to the perfectly-periodic scheduling problem is presented — *the multiple-sizes multiple-servers model*. By presenting a *lower-bound* on the optimal schedule we show (in Chapter 3) that the multiple-sizes model is inherently different from the unit-size model. The lower-bound also shows the importance of the extent when trying to analyze the multiple-sizes model. We then proceed by introducing the *balancing* technique (in Chapter 4) and the algorithms it produces (in Chapters 4 and 5). We generalize the schedules produced by balancing and present the notion of *separable schedules* (in Chapter 6). We show that there are operations we can perform on such schedules. We conclude by showing (in Chapter 7) an algorithm that uses all of the techniques above to achieve a good approximation factor and show that it is easily generalized to the multiple-servers model. Figure 8.1 summarizes all of the algorithms presented throughout this work and the techniques used to obtain them.

8.1 Open Problems

The extended model and the algorithms presented in this work give rise to related problems that require further research.

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 to the optimal schedule of the given instance instead of giving an approximation ratio relative to the bandwidth.

2. **Dynamic model.** All of the algorithms presented above function in a *static model*: we assume that the set of input jobs does not change after the schedule is constructed. We might want to consider a *dynamic model* where jobs can be

Name	Input	Output	Techniques used
bal (fig. 4.1)	J *s.t.* $\tau_i = c \cdot 2^{e_i}$, $\Delta \geq R \cdot \frac{t}{t^*}$, parameter $t^* = t/2^e$.	S *s.t.* $\tau_i^S \leq \tau_i$.	Balancing.
s&b (fig. 4.3)	J *s.t.* $\tau_i = c \cdot 2^{e_i}$, parameter $t^* = t/2^e$.	S *s.t.* $\tau_i^S \leq (\beta + R \cdot \frac{t}{t^*})\tau_i$.	Scaling periods by a common factor and balancing.
A (fig. 4.4)	J.	S *s.t.* $C_{AVE}(J,S) \leq \frac{9}{8} + \frac{3}{2}R + \frac{1}{2}R^2$, $C_{MAX}(J,S) \leq 2(1+R)$.	Rounding to next power of 2 and running s&b.
B (fig. 5.1)	J.	S *s.t.* $C_{MAX}(J,S) \leq 1 + \frac{\sqrt{2}}{2} + 2R$.	Rounding to next power of 2, rounding $\frac{\tau_i}{\sqrt{2}}$ to next power of 2, s&b.
P (fig. 6.1)	Separable S with bins w, parameter p.	Separable S' with bins $\frac{w}{p} + B$ *s.t.* $\tau_i^{S'} \leq (1 + \frac{pB}{w})\tau_i^S$.	Separable schedules.
M (fig. 6.3)	Separable $S_1, \ldots S_k$ with bins $w_1, \ldots w_k$.	Periodic S *s.t.* $\tau_i^S \leq \frac{\sum_{\ell=1}^k w_\ell}{w_\ell}\tau_i^{S_\ell}$.	Separable schedules.
C (fig. 7.1)	J, parameters k, L.	S *s.t.* $C_{MAX}(J,S) \leq \left(1 + \frac{1}{k}\right)\left(1 + \frac{1}{L}\right) \cdot (1 + 2k(L+1)R)$.	Rounding to next power of $2^{1/k}$, s&b, separable schedules (P, M).
genP (fig. 7.2)	Separable S with bins w, parameters m, p *s.t.* $m \mid p$.	Separable $S_1, \ldots, S_m$ with bins $\frac{w}{p} + B$ *s.t.* $\tau_i^{S_\ell} \leq \frac{1}{m}(1 + \frac{pB}{w})\tau_i^S$.	Separable schedules, parallelizing bins.
genC (sec. 7.2)	J (m machines), parameters k, L.	S *s.t.* $C_{MAX}(J,S) \leq \left(1 + \frac{1}{k}\right)\left(1 + \frac{1}{L}\right) \cdot \left(1 + 2k\left(L + \frac{1}{m}\right)R\right)$.	Rounding to next power of $2^{1/k}$, s&b, separable schedules and parallelizing (genP, M).

Figure 8.1: Summary of presented algorithms and techniques.

added or taken off in an online manner. This question is relevant in many practical settings.

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 keeping the representation of the schedule on the server small is known as the *dispatching problem*. A *dispatching scheme* for perfectly periodic schedules represented in special "tree schedules" is presented in [8]. We would like to construct good dispatching schemes for the schedules produced by the algorithms presented above.

Bibliography

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

[2] S. Acharya, R. Alonso, M. J. Franklin, and S. B. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proc. 1995 ACM SIGMOD*, pages 199–210, 1995.

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

[4] S. Anily, C. A. Glass, and R. Hassin. Scheduling of maintenance services to three machines. *Annals of Operations Research*, 86:375–391, 1999.

[5] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation cost of periodic scheduling. In *Proc. of the 9th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 11–20, 1998.

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[7] A. Bar-Noy, V. Dreizin, and B. Patt-Shamir. Efficient Periodic Scheduling by Trees. To appear in *INFOCOM 2002*, June 2002.

[8] Z. Brakeski, V. Dreizin, and B. Patt-Shamir. Dispatching in Perfectly-Periodic Schedules. Unpublished manuscript, 2001.

[9] Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queueing theory. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 376–385, 1996.

[10] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir. Nearly optimal perfectly-periodic schedules. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 107–116, 2001.

[11] Y. D. Chung and M.-H. Kim. QEM: A scheduling method for wireless broadcast data. In *Proc. Sixth International Conf. on Database Systems for Advanced Applications*, pages 135–142. IEEE Computer Society, 1999.

[12] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Energy efficient indexing on air. In R. T. Snodgrass and M. Winslett, editors, *Proc. 1994 ACM SIGMOD*, pages 25–36. ACM Press, 1994.

[13] C. Kenyon and N. Schabanel. The data broadcast problem with non-uniform transmission times. In *Proc. 10th SODA*, pages 547–556, Jan 1999.

[14] C. Kenyon, N. Schabanel, and N. Young. Polynomial-time approximation scheme for data broadcast. In *Proc. 32nd STOC*, pages 659–666, May 2000.

[15] S. Khanna and S. Zhou. On indexed data broadcast. In *Proc. 30th ACM STOC*, pages 463–472, 1998.

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

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

[18] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proc. First Symposium on Operating Systems Design and Implementation*, November 1994.

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