On Distributed Smooth Scheduling

Dedicated to the memory of Professor Shimon Even for his inspiration and encouragement

Ami Litman* Shiri Moran-Schein*

Abstract

This paper studies evenly distributed sets of natural numbers and their applications to scheduling in a distributed environment. Such sets, called *smooth sets*, have the property that their quantity within each interval is proportional to the size of the interval, up to a bounded additive deviation; namely, for $\rho, \Delta \in \mathbb{R}$ a set A of natural numbers is (ρ, Δ)-*smooth* if $\mathrm{abs}(|I| \cdot \rho - |I \cap A|) < \Delta$ for any interval $I \subset \mathbb{N}$.

The current paper studies scheduling *persistent clients* on a single slot-oriented *resource* in a flexible, predictable and distributed manner. Each client γ has a given *rate* ρ_γ that defines the share of the resource he is entitled to receive and the goal is a *smooth schedule* in which, for some predefined Δ, each client γ is served in a (ρ_γ, Δ)-smooth set of slots (natural numbers). The paper focuses on a *distributed environment* where each client by itself (without any inter-client communication) *resolves* (computes), slot after slot, whether or not it owns this slot. The paper presents extremely efficient schedules under which a client resolves each slot in a constant time.

The paper considers two scheduling frameworks. The first one, the *Flat Scheduling Framework*, is the common problem where the rates of the clients are given a priori. In the second and novel framework, the *Open-Market Scheduling Framework*, fractions of the resource are bought and sold by *dealers*. Each dealer, upon receiving his set of slots, may choose either to become a client and use his share, or to remain a dealer and sell fractions of his share to other dealers. In this framework, the allocation process is highly distributed; moreover, fractions of several resources can be combined into a single virtual resource of new capabilities.

The paper presents two scheduling techniques. Both techniques, in both frameworks, produce smooth schedules with highly efficient distributed resolutions — a client resolves each slot in $O(1)$ time on a RAM with a moderate number of memory words, all of a small size. Each technique has its pros and cons. For example, one technique utilizes 100% of the resource but its resolution algorithm requires a number of words which is linear in the number of clients; the other technique utilizes only 99% of the resource but its resolution algorithm requires just $O(1)$ words.

One of these techniques yields a solution to Tijdeman's Hierarchial Chairman Assignment Problem which outperforms prior solutions. The other technique naturally extends to the problem of scheduling multiple resources, under the restriction that a client may be served concurrently by at most one resource. The extension yields the first solution to this problem having efficient distributed resolution. Prior solutions produce a special type of smooth scheduling called *P-fair scheduling*, are centralized, and are less efficient than ours.

Keywords: Persistent scheduling, Smooth scheduling, Distributed Resolution, Scheduling of multiple resources, P-fair Scheduling

*Department of Computer Science, Technion, Haifa 32000, Israel. E-mail: {litman,mshiri}@cs.technion.ac.il.

1 Introduction

1.1 Smooth and Distributed Scheduling

This paper studies evenly distributed sets of natural numbers and their applications to scheduling in a distributed environment. Such sets, called *smooth sets*, have the property that their quantity within each interval is proportional to the size of the interval, up to a bounded additive deviation; namely, for $\rho, \Delta \in \mathbb{R}$ a set A of natural numbers is (ρ, Δ)-*smooth* if $\text{abs}(|I| \cdot \rho - |I \cap A|) < \Delta$ for any interval $I \subset \mathbb{N}$; a set A is Δ-smooth if it is (ρ, Δ)-smooth for some real number ρ. An earlier paper of us [27] establishes the concept of smooth sets and the current paper builds on the mathematical infrastructure constructed there. An accompanying paper [26] studies applications of smooth sets to scheduling in a centralized environment.

As demonstrated in this paper and in [26], smooth sets are highly attractive for scheduling *persistent clients* [10] having pre-defined rates on a single slot-oriented *resource* in a flexible and predictable manner. In this framework, time is divided into discrete *slots* and in each slot the resource (e.g., a broadcast channel) may serve at most one *client* (e.g., a transmitter). Each client γ has a pre-defined *rate* $\rho_\gamma \in [0, 1]$ that defines the share of the resource he is entitled to receive. A *smooth schedule* for such a problem is a schedule in which, for some predefined Δ, each client γ is served in a (ρ_γ, Δ)-smooth set of slots (natural numbers). Such a schedule enjoys the following two attractive properties. **Proper rate** — the average amount of service received by a client γ in the long-run is a ρ_γ fraction of the resource. **Bounded deviation** — the number of slots a client γ receives during any k consecutive slots deviates from his nominal share of $k \cdot \rho_\gamma$ by less than a pre-specified constant.

We apply a novel approach to scheduling in which the scheduling process is divided into two stages. In the *allocation stage* each client is allocated an infinite set of slots that is generated via abstract mathematical operations. In the *online stage* an algorithm computes, slot after slot, the client which is served in this slot. This contrast most previous approaches in which the allocation is just a by-product of the online algorithm. Moreover, the mathematical operations we use enable exceedingly fast online algorithms working in a constant time per slot. (The allocation stage is also efficient and takes polynomial time.) This contrast most other algorithms that produce smooth schedules (e.g., [12, 13, 30]) as they are based on priority queues and their time per slot is at least logarithmic in the size of the queues.

Most of the work on scheduling persistent clients, including our accompanying paper [26], consider a (single) centralized online algorithm that computes, slot after slot, the owner of the slot. In contrast, this paper focuses on **distributed resolution** of the allocation in which each client independently (without any inter-client communication) *resolves* (computes), slot after slot, whether or not he owns the slot.

This paper presents two scheduling techniques; the first technique has a resolution algorithm of $O(1)$ time per slot on a RAM having $O(n)$ memory words when n is the number of clients; the second technique has a resolution algorithm of $O(1)$ time per slot on a RAM having $O(1)$ memory words; the words of both RAMs are of a small[1] width. The allocated set of slots[2] and their resolvers are also generated efficiently, in polynomial time.

Our first technique yields a solution to Tijdeman's Hierarchial Chairman Assignment Problem which outperforms prior solutions (Section 2). The other technique naturally extends to the problem of scheduling multiple resources, under the restriction that a client may be served concurrently by at most one resource (Section 1.4). The extension yields the first solution to this problem having efficient distributed resolution — it resolves each slot in $O(1)$ time per slot on a RAM having $O(1)$ memory words. Prior algorithms for this problem [13, 12] produce a special type of smooth schedule called *P-fair schedule* [12]. These algorithms are centralized, and the fastest one [13] works in $O(m \log n)$ time per slot on a RAM having $O(n)$ memory words, where n is the number of clients and m is the number of resources.

Finally, our work demonstrates that smooth schedules are highly attractive for scheduling communication links (Section 7) and, in particular, for scheduling the links of a connection-oriented packet

[1]The bounds on the number of memory words and their width are critical; without one of them it is a simple matter to encode the entire schedule in a finite number of words and to retrieve the resolution of the slots, one by one, in constant time.

[2]What is generated in polynomial time is, of course, not an infinite set of slots, but a finite encoding of such a set.

switching network [26]. It also demonstrates that in many applications the difference between a 2-smooth schedule (as in [12]) and, for example, an 11-smooth schedule (as some of the schedules produced here) is not essential.

1.2 Scheduling Frameworks

We henceforth identify the slots of the resource with the natural numbers and use the following notations. For a set A of slots (natural numbers) if the limit $\lim_{|I|\to\infty}(|A\cap I|/|I|)$ exists where I ranges over the intervals of natural numbers[3], then this limit is called the *rate* of A. Note that a (ρ,Δ)-smooth set has rate ρ. For a finite set of clients Γ, a Γ-*allocation* is a system $\{A_\gamma \mid \gamma\in\Gamma\}$ of disjoint sets of natural numbers. Such an allocation is Δ-*smooth* if every A_γ is Δ-smooth; the allocation is *smooth* if it is Δ-smooth for some Δ.

We consider two scheduling frameworks. In the most rudimentary framework, the *Flat Scheduling Framework*, there is a set of clients, each with a predefined rate. The goal is a smooth allocation combined with an efficient resolution algorithm for each allocated set.

The second and novel *Open-Market Framework* is based on the paradigm of an open-market in which fractions of the resource are bought and sold by *dealers*. The original owner of the resource sells fractions of it to several dealers. Each dealer again sells fractions of his share, and so on. Each dealer has a predefined fraction (rate) of the resource he wants to buy and it is implicity required that the sum of the rates of the dealers buying from a certain dealer does not exceed the rate of that dealer. Upon buying his share each dealer receives a subset of the slots having the desired rate. (This subset is independent of what the dealer does later with his slots.) The dealer then may choose either to become a client and use his share, or to remain a dealer and sell his share to other dealers. In this framework the allocation process is highly distributed; moreover, fractions of several resources can be combined into a single virtual resource of new capabilities, as illustrated in Section 7.

1.3 Scheduling Techniques

This work presents two scheduling techniques: the *recursive technique* and the *intervals technique*. Both techniques have extremely efficient resolution algorithms of $O(1)$ time per slot.

The recursive technique This technique works as follows. The slots of the resource are divided, in a certain way, into several smooth sets having certain rates. With regard to scheduling, a resource is just an infinite sequence of slots; hence, each such set constitutes a virtual resource which is again recursively divided into sets, and so on.

The recursive technique in the flat scheduling context is summarized by the following theorem, which uses the following terminology: A *k-bit fraction* is a number of the form $l/2^k$ for some $l\in\mathbb{N}$ and a *binary fraction* is a number that is a k-bit fraction for some k. A *k-bit RAM* is a RAM having k-bit words. (Such a RAM performs standard operations on these words in constant time.)

Theorem 2: Let $\{\rho_\gamma \mid \gamma\in\Gamma\}$ be an instance of the flat scheduling problem s.t. $\Sigma_{\gamma\in\Gamma}\ \rho_\gamma\le 1$. Then there is a 4-smooth allocation $\{A_\gamma \mid \gamma\in\Gamma\}$ s.t. each A_γ has rate ρ_γ. Moreover[4], if all the ρ_γ are k-bit fractions then each A_γ has a resolver of $O(|\Gamma|)$ memory words and $O(1)$ time per slot on any $\Omega(k)$-bit RAM.

(The numbering of this theorem, as well as of the other theorems stated in the introduction, follows that of the main part of the paper.) The binary base in the above theorem is not mandatory; e.g., the last requirement, that the rates are k-bit fractions, can be replaced by the requirement that the rates are k-digit decimal fractions.

In the open-market context the recursive technique has the additional *big-portion* restriction: A dealer may not sell a very large fraction of his share to any single buyer. To be specific, the α-*portion restriction* is the restriction that a dealer does not sell a fraction greater than α to any single buyer. We present the recursive technique with $\alpha = 2/3$ and the following theorem is a weak version of Theorem 3 that summarizes the recursive technique in the open-market context.

[3]That is, there is a real number ρ s.t. for any $\epsilon>0$, for all sufficiently large intervals I, we have $\mathrm{abs}(|A\cap I|/|I|-\rho)<\epsilon$.

[4]The theorem without this last addition is already known; in fact, Tijdeman [30] has shown that there is such a 2-smooth allocation; more on Tijdeman's work is discussed in Section 2.

Theorem 3': In the open-market context and under the 2/3-portion restriction the recursive technique assigns to any dealer (or client) γ a 6-smooth set A_γ with the appropriate rate. Moreover, if the rates of all the dealers and all the clients are k-bit fractions then A_γ can be resolved in $O(1)$ time per slot using $O(N)$ memory words on any $\Omega(k)$-bit RAM, where N is the total number of dealers and clients.

The above numbers 2/3 and 6 are interdependent; for a general $\alpha \in [0, 1)$ the recursive technique under the α-portion restriction produces a $O(1/(1-\alpha))$-smooth allocation.
The Intervals Technique This technique applies a reduction from allocation of slots to allocation of sub-intervals of the unit interval; namely, it is based on a one-to-one partial function from the unit interval into the natural numbers s.t. an interval of length ρ is mapped onto a subset of the natural numbers with rate ρ. Thus, this technique allocates disjoint sets of slots by allocating disjoint real intervals and applying the above function on these intervals. The following theorem summarizes the intervals technique in the flat scheduling context. In this theorem, a number is a *k-bit rational* if it is the ratio of two k bit numbers.

Theorem 7: Let $\{\rho_\gamma \mid \gamma \in \Gamma\}$ be an instance of the flat scheduling problem s.t. $\Sigma_{\gamma\in\Gamma}\ \rho_\gamma \leq 0.99$. Then there is an 11-smooth allocation $\{A_\gamma \mid \gamma \in \Gamma\}$ s.t. each A_γ has rate ρ_γ. Moreover, if ρ_γ is a k-bit rational then A_γ has a resolver of $O(1)$ memory words and $O(1)$ time per slot on any $\Omega(k)$-bit RAM.

In the open-market context the only restriction on the rates of the buyers is that their sum is not greater than the rate of the seller. Unfortunately, the set of slots allocated to a dealer is not necessarily smooth. (In the recursive technique these sets are smooth.) However, any such set with rate ρ' has an 11-smooth subset with rate $0.99 \cdot \rho'$. Thus, a dealer who wants to become a client with rate ρ should buy a set with rate $\rho' = (1/0.99) \cdot \rho$ and use a 0.99 fraction of this set. We henceforth refer to such ρ and ρ' as the *net rate* and the *gross rate* of the buyer. Dealers buy and sell sets having the appropriate gross rates while clients use sets having the appropriate net rate. This is summarized by the following theorem.

Theorem 6: In the open-market context the intervals technique allocates to any dealer γ a set A_γ with the appropriate gross rate. When γ becomes a client he is allocated an 11-smooth set $A'_\gamma \subset A_\gamma$ with the appropriate net rate. Moreover, if the net rate of γ is a k-bit rational then A'_γ can be resolved in $O(1)$ memory words and $O(1)$ time per slot on any $\Omega(k)$-bit RAM.

In both frameworks the above two numbers 0.99 and 11 are interdependent; in fact, the number 0.99 can be replaced with any $1-\epsilon$, where $\epsilon > 0$; in this case the allocated sets are $O(\log(1/\epsilon))$-smooth. In the intervals technique (in both frameworks) the dealers/clients are highly insensitive to each other, as expressed by the following two properties. A client with a rational rate can resolve his set of slots in $O(1)$ time per slot using $O(1)$ memory words on a RAM whose width depends only on the rate of this (and no other) client. The second property, which is not mentioned by the above theorems, is that the technique enables is *piecewise allocation* of slots; that is, a buyer/client can receive his set of slots on his arrival, independently of any later buyers. This implies that in the flat scheduling framework the set allocated to a client depends only on the preceding clients; in the open-market framework the set of a buyer depends only on the set of its seller and on the rates of the previous buyers from that seller. Actually, the dependency on the sets of the previous buyers/clients is very restricted — only the sum of their rates (and not the manner this sum is divided into rates) is relevant. The recursive technique lacks these properties.

1.4 Multiple Resources

A natural extension of our work concerns scheduling of m identical resources where the sum of the rates of the clients is (of course) at most m and the rate of each client is at most one. If it is permitted to serve a single client by several resources in one slot then it is easy to transform any scheduling technique of a single resource into a scheduling technique of several identical resources, as indicated in [12]; see also Subsection 6.3.

However, in some applications (e.g., CPU scheduling) a client may be served concurrently by at most one resource. As pointed out by Liu [28] and again by Baruah et al. [12] "the simple fact that a client can use only one resource even when several resources are free at the same time adds a surprising amount of difficulty" (to the scheduling problem). Surprisingly, it turns out that in our intervals technique this is not the case. Namely, a straightforward extension of the technique provides allocation having the same smoothness and efficient resolvers, as well as other properties, of the single resource case.

This extension yields the first smooth schedule with efficient distributed resolvers for the above problem and these resolvers are of $O(1)$ time per slot and $O(1)$ memory words. Prior algorithms for this problem [13, 12] produce smooth schedules of a special type called P-fair schedules [12], are centralized, and the fastest one [13] works in $O(m \log n)$ time per slot and $O(n)$ memory words, where n is the number of clients and m is the number of resources. P-fair scheduling is further discussed in Section 2.

The rest of this paper is organized as follows. Section 2 presents related work, Sections 3 and 5 review notations and lemmas from [27] which are used in the recursive technique and in the intervals technique, respectively. Sections 4 and 6 establish the recursive technique and the intervals technique, respectively, and Section 7 presents a scheduling application that demonstrates the advantages of our approach.

2 Related Work

Our work is related to various works on assignment and scheduling problems, as follows.

The Chairman Assignment Problem. An early scheduling problem, due to Tijdeman [30], is the following *Chairman Assignment Problem*. There is a union of several states, each having a positive weight. Every year a union chairman has to be selected in "such a manner that no state has a good reason to complain". Casting this problem into the framework of our work, the objective is that the rate of the set of years assigned to a state is proportional to its weight and is Δ-smooth for a Δ which is as small as possible. This problem is a special case of our flat allocation problem in which the sum of the rates is one and all the slots are required to be allocated. Tijdeman [30] proved that any such problem has a solution with a 2-smooth[5] allocation. Moreover, an algorithm of $O(\log n)$ time per slot, where n is the number of states, is implicit in his solution. Later and similar solutions are implicit in [16, 12]. Baruah et al. [14] studied a *dynamic* variant of the problem, in which clients (states) may leave and join the system (union), and thus change the rates of the other clients.

Tijdeman, in the same paper [30], has presented the *Hierarchial Chairman Assignment Problem* which is the following generalization of the above problem. There is an hierarchial grouping of states into organizations. That is, states are grouped into unions, unions are grouped into federations, and so on. The overall organization has a yearly assigned chairman, and this assignment has to be smooth, not only w.r.t. the states, but also w.r.t. any organization (union, federation, etc.). Tijdeman [30] has presented a solution of this problem in which $\Delta = l+1$, where l is the depth of the hierarchial structure of the organizations.

Our recursive technique, in the open-market context, solves the following variant of the Hierarchial Chairman Assignment Problem. Our variant is somewhat weaker and somewhat stronger than Tijdeman's problem. It is stronger since we require the set of years (slots) assigned to an organization (dealer) to be independent of the manner this organization is divided into sub-organizations. Our problem is weaker since we require the grouping to obey a reasonable big-portion restriction; for example, we may require that no sub-organization of an organization has more than 90% of the latter rate. Our recursive technique establishes the following theorem.

Theorem 4: Any instance of the Hierarchial Chairman Assignment Problem obeying an α-portion restriction for $\alpha \in (0, 1)$ has an allocation in which any organization is allocated an $O(1/(1-\alpha))$-smooth set with the appropriate rate. $\diamond$

To the best of our knowledge, this is the first solution to the Hierarchial Chairman Assignment Problem in which the deviation is independent of the depth of the hierarchial structure of the organizations.

Fast Distributed Resolution. To the best of our knowledge, the only previously known schedules with distributed resolutions of $O(1)$ time per slot are the *perfectly-periodic schedules* introduced by Bar-Noy, Nisgav and Patt-Shamir [11] and further studied in [17, 9]. Their approach takes to the

[5] Tijdeman measured smoothness in a slightly different manner than ours; we cast his results into our terminology.

extreme the concept of 'as evenly distributed as possible'; namely, the time-slots allocated to a client are required to constitute an arithmetic sequence. Hence, the rate of each client is required to be $1/p$ for some $p \in \mathbb{N}$. This approach is extremely inflexible: not only the rates themselves are highly limited, but many combinations of rates with sum less than one, such as the pair $(1/2, 1/3)$, are impossible to schedule. Our work shows that fast and distributed resolution can be achieved with almost no restrictions on the rates.

Periodic Scheduling. The *periodic scheduling problem*, e.g [24, 22, 15], addresses the issue of CPU scheduling, as follows. There are n *tasks* (clients) and each task γ is associated with two natural numbers $e_\gamma \leq p_\gamma$. The schedule has to allocate the CPU to a task γ for exactly e_γ time-slots in each interval of the form $[p_\gamma k, p_\gamma(k+1))$ for $k \in \mathbb{N}$. This framework, which is intended for CPU scheduling, is unsuitable for many other applications since it may produce highly unsmooth schedules, as demonstrated in Section 7.

P-fair Scheduling. A variant of the periodic scheduling problem which implies smooth scheduling is the *P-fair Scheduling Problem* introduced by Baruah et al. [12] and further studied in [13, 2, 29, 21, 19, 5, 3, 4, 6, 18]. In this variant there are m identical resources (rather than just a single one) and a task may be served concurrently by at most one resource. The variant requires that, for each t, during the first t slots each task γ is scheduled either $\lceil p_\gamma/e_\gamma \cdot t\rceil$ or $\lfloor p_\gamma/e_\gamma \cdot t\rfloor$ slots; such a schedule is 2-smooth [27]. Note that in the case of a single resource this problem is essentially the (non-hierarchial) Chairman Assignment Problem mentioned above. Moreover, the P-fair scheduling problem is essentially a restricted variant of the multiple resources version of our flat scheduling problem; in this variant the only restriction on the rates of the clients is that their sum is at most 1 and the scheduling is required to be 2-smooth.

All the works on P-fair scheduling consider centralized scheduling algorithms and the currently fastest one, due to Baruah et al. [13], works in $O(m \log n)$ time per slot on a RAM with $O(n)$ memory words of a small width. In this paper we solve a scheduling problem that differs from the P-fair problem in three main aspects: Our online scheduling algorithm is distributed rather than centralized. The sum of the rates of the clients must not exceed $0.99 \cdot m$ (rather than m) and the allocation is 11-smooth (rather than 2-smooth or P-fair). In many applications, however, the difference between a 2-smooth schedule and an 11-smooth schedule is not essential. As said, our distributed online algorithm is exceedingly fast and each client resolves each slot in $O(1)$ time on a RAM of $O(1)$ memory words of width k, where the rate of the client in question is a k-bit rational. Furthermore, our technique works also in the novel open-market framework in which the allocation process is highly distributed.

Our accompanying paper on centralized scheduling [26] solves another variant of the P-fair problem. Again, this variant differs from the P-fair problem in requiring that the sum of the rates is at most $0.99 \cdot m$ and in producing a 10-smooth allocation. Our algorithm works in, essentially, $O(m)$ time per slot on a RAM with $O(n)$ memory words of a small width. This is a significant improvement to the prior fastest such algorithm [13] that works in $O(m \log n)$ time per slot on a RAM with $O(n)$ memory words. Moreover, that paper [26] introduces a novel approach in which each resource independently computes, slot after slot, which client to serve in this slot. The paper presents such an algorithm that works in, essentially, $O(1)$ time per slot on a RAM with $O(n)$ memory words of a small width.

The Broadcast Disk Problem. Another problem that is somewhat similar to our work is the *Broadcast Disk Problem* [1, 8, 23]. In this framework there is a database of m *pages* (clients) each associated with the probability that a random user wishes to access the page. The resource in this case is a *broadcast channel* that in each slot can broadcast at most M pages. The goal is to schedule the broadcast of the pages in such a way that minimizes the average waiting time of the random user.

Although this problem seems somewhat similar to ours scheduling problems, it comes out that the concept of rate, which is critical in our work, is completely absent inthe broadcast problem. Namely, an instance of this problem can have two optimal schedules in which the rates of the same page are different and can have a third optimal solution in which this page does not have a rate. For example, consider the case of $m = 2$, $M = 1$ and the two probabilities of the pages a and b are $3/4$ and $1/4$. Following arguments of Anily et al. [7], each of the following two periodic schedules are optimal: $ababab\cdots$ and $aabaabaab\cdots$. The rate of b is $1/2$ in the first schedule and $1/3$ in the second one. Its not hard to "mix" these two solutions into a third optimal solution in which the set of slots allocated to b has no rate.

3 Imported Tools - Part 1

As said, this paper builds on the mathematical infrastructure of smooth sets developed in [27]. This section reviews notations and lemmas from there which are used by the recursive technique; Section 5 reviews notations and lemmas which are used by the intervals technique. Full proofs and elaborations are available in [27].

Smooth Sets. Recall that a set $A \subset \mathbb{N}$ is (ρ, Δ)-*smooth* if $\text{abs}(|I| \cdot \rho - |I \cap A|) < \Delta$ for any finite interval I of $\mathbb{N}$; a set is Δ-smooth if it is (ρ, Δ)-smooth for some ρ.

Lemma 1: [27] Let $A, B \subset \mathbb{N}$ be (ρ_1, Δ_1)-smooth and (ρ_2, Δ_2)-smooth, respectively. Then:

a. If $A \cap B = \emptyset$ then $A \cup B$ is $(\rho_1 + \rho_2, \Delta_1 + \Delta_2)$-smooth.

b. The set $\mathbb{N} \setminus A$ is $(1 - \rho_1, \Delta_1)$-smooth.

c. If $B \subset A$ then $A \setminus B$ is $(\rho_1 - \rho_2, \Delta_1 + \Delta_2)$-smooth.

Lemma 2: [27, 25] A $(\rho, 1)$-smooth set $A \subset \mathbb{N}$ exists for any $0 \leq \rho \leq 1$.

The following two lemmas concern 1-smooth sets; the first one provides an arithmetic characterization of the 1-smooth sets and the second one shows that any 1-smooth set with a rational rate can be resolved efficiently, in constant time. To that end, for $x \in \mathbb{R}$, define $B'_{\rho,x} \triangleq \lfloor \mathbb{Z} \cdot (1/\rho) + x \rfloor \cap \mathbb{N}$ and $B''_{\rho,x} \triangleq \lceil \mathbb{Z} \cdot (1/\rho) + x \rceil \cap \mathbb{N}$.

Lemma 3: [27] Let $0 < \rho \leq 1$. Then:

a. $B'_{\rho,x}$ is $(\rho, 1)$-smooth for any x.

b. $B''_{\rho,x}$ is $(\rho, 1)$-smooth for any x.

c. Any $(\rho, 1)$-smooth set is either of the form of Statement (a) or of the form of Statement (b).

Lemma 4: [27] Let B be $(\rho, 1)$-smooth and let $0 < \rho = p/q$ and $p, q \in \mathbb{N}$. Then $B = B'_{\rho,i'/p} = B''_{\rho,i''/p}$ for some integers $0 \leq i', i'' < q$.

Composition of Sets. The recursive technique is based on set composition, defined as follows. For $A, B \subset \mathbb{N}$ the set composition of A and B is the subset D of A s.t., for all i, the i-th member of A is in D iff the i-th member of $\mathbb{N}$ is in B and A has an i-th member. In other words, the *composition* of A and B is $A \circ B \triangleq \{n \in A \ : \ |[0, n) \cap A| \in B\}$. Note that for any $D \subset A$ there is a set $B \subset \mathbb{N}$ s.t. $D = A \circ B$ and, when A is infinite, this B is unique.

Lemma 5: [27] Composition of sets is an associative operator.

The following lemma shows that the composition of two smooth sets is smooth and moreover, under some restrictions, $A \circ B$ is as smooth as A.

Lemma 6: [27] Let A and B be (ρ_1, Δ_1)-smooth and (ρ_2, Δ_2)-smooth subsets of $\mathbb{N}$. Then:

a. $A \circ B$ is $(\rho_1 \cdot \rho_2, \rho_2 \cdot \Delta_1 + \Delta_2)$-smooth.

b. If $\Delta_2 = 1$ and $\rho_2 \leq (\Delta_1 - 1)/\Delta_1$ then $A \circ B$ is $(\rho_1 \cdot \rho_2, \Delta_1)$-smooth.

Given a (ρ, Δ)-smooth set A and two positive numbers β_1 and β_2 with $\beta_1 + \beta_2 = 1$, it is sometimes desirable to partition A into two sets A_1 and A_2 s.t. each A_i is $(\rho \cdot \beta_i, \Delta)$-smooth. The following theorem, which is the cornerstone of our recursive technique, provides such a partition under some restrictions on β_1 and β_2.

Theorem 1: [27] Let $A \subseteq \mathbb{N}$ be (ρ, Δ)-smooth and let $\beta_1 + \beta_2 = 1$ with $\beta_1, \beta_2 \leq (\Delta - 1)/\Delta$. Then there is a partition of A into $\langle A_1, A_2 \rangle$ s.t. each A_i is $(\beta_i \cdot \rho, \Delta)$-smooth. In fact, for any partition of $\mathbb{N}$ into $\langle B_1, B_2 \rangle$ where each B_i is $(\beta_i, 1)$-smooth, the sets $A_1 = A \circ B_1$ and $A_2 = A \circ B_2$ satisfy the above requirement.

The above requirement '$\beta_1, \beta_2 \leq (\Delta - 1)/\Delta$' is mandatory, as stated by the next lemma.

Lemma 7: [27] Let $0 < \rho < 1$ and $\Delta \geq 1$. Then there is a (ρ, Δ)-smooth set A s.t. for any $\beta < 1$ which is close enough to 1 there is no subset of A which is $(\rho \cdot \beta, \Delta)$-smooth.

4 The Recursive Technique

This section presents our first scheduling technique, the *recursive technique*, which is based on composition of smooth sets and works as follows. The slots of the resource are divided, in a certain way, into several smooth sets having certain rates. With regard to scheduling, a resource is just an infinite sequence of slots. Hence, each such set constitutes a virtual resource which is again recursively divided into sets, and so on.

In both frameworks this technique, as the other scheduling technique, produces a smooth schedule having highly efficient resolution algorithms of $O(1)$ time per slot on a RAM with a moderate number of words of a small width. This $O(1)$ resolution requires that the *binary-fraction restriction* is met — all rates are of the form $l/2^j$ for some $l, j \in \mathbb{N}$. In addition, this technique also has an efficient polynomial allocation algorithm and the resolution programs are also generated in polynomial time.

In the flat framework the technique produces a 4-smooth allocation. In the open-market framework the technique produces a 6-smooth allocation, while having the additional 2/3-portion restriction — a dealer can not sell a portion greater than 2/3 of his share to any single buyer. (The numbers 2/3 and 6 are interdependent.)

This section is organized as follows. Subsection 4.1 studies resolution of composed sets and Subsections 4.2 and 4.3 present the recursive technique in the flat framework and in the open-market framework, respectively. The recursive technique, without the issue of fast resolution, is quite simple. Therefore, it is recommended to read the rest of this section in two passes. The first pass should ignore all the claims and arguments related to resolution and, in particular, should skip Subsection 4.1. This pass suffices to establish our solution to the Tijdeman's Hierarchial Chairman Assignment Problem.

4.1 Resolution of Composed Sets

Recall that a resolution algorithm of a set of slots (natural numbers) is an algorithm that resolves, slot after slot, whether the slot belongs or does not belong to the set. This subsection studies the subject of resolving composed sets and, in particular, how to combine two resolution algorithms of sets A and B into a resolution algorithm for the set $A \circ B$.

We assume that the resolution algorithm is performed on the popular *RAM* model of computation [20]. Specifically, the RAM memory is organized in binary *words* of k bits, each having a distinct k-bit address. Such a RAM is called a RAM of *width* k or a k*-bit RAM*. Each of the following operations takes one time unit (step) on such a RAM: memory access; the basic arithmetic operations on words (addition, subtraction, multiplication and division, all modulo 2^k); the basic logical operations on words ($\neg, \wedge$ and $\vee$); comparing two words and branching accordingly; and calling a subroutine.

We refer to a resolution algorithm of a set A of slots as a *resolver* of A. A resolver operates on a RAM of a predefined width; such a resolver is of *space* s and *time* t if it uses at most s memory words and resolves each slot in at most t time units. Recall that a number is a k*-bit rational* if it is the ratio of two k-bit numbers. The following lemma is implied by Lemma 4.

Lemma 8: Any 1-smooth set whose rate is a k-bit rational has a resolver of $O(1)$ time and $O(1)$ space on any $\Omega(k)$-bit RAM.

Let us fix ζ to be a number, provided by the previous lemma, s.t. any 1-smooth set whose rate is a k-bit rational has a resolver of ζ time and $O(1)$ space on the k-bit RAM.

To facilitate resolution of composed sets, we assume that a resolver is implemented as follows. It is a subroutine that is called with no arguments and returns either one or zero, denoting whether the next slot belongs or does not belong to the set it resolves. The resolver uses a subroutine, called ONE, that returns the constant value 1. A resolver returns 1 **only** by calling, directly or indirectly, that subroutine and returning the value provided by the subroutine; the resolver use this ONE subroutine only in the above manner. It follows from the definition of composition of sets that given a resolver R_{A_1} of $A_1 \subset \mathbb{N}$ and a resolver R_{A_2} of $A_2 \subset \mathbb{N}$, a resolver of $A_1 \circ A_2$ is obtained by replacing every call to ONE in R_{A_1} with a call to R_{A_2}; this resolver is referred to as $R_{A_1} \circ R_{A_2}$. The following lemma is straightforward.

Lemma 9: Consider a RAM of a fixed width. Let $A_1, A_2 \subset \mathbb{N}$ have resolvers R_{A_1} of t_1 time and s_1 space and R_{A_2} of t_2 time and s_2 space, respectively. Then $R_{A_1} \circ R_{A_2}$ resolves $A_1 \circ A_2$ in $t_1 + t_2$ time and $s_1 + s_2$ space.

A resolver is of *amortized time* $\langle t, l\rangle$ or, in short, of $\langle t, l\rangle$ *time*, if any k consecutive calls to the resolver consume at most $t \cdot (k + l)$ time. For a set $A \subset \mathbb{N}$, let rate(A) denote the rate of A.

Lemma 10: Consider a RAM of a fixed width. Let $A_1 \subset \mathbb{N}$ be a Δ-smooth set having a resolver R_{A_1} of $\langle t_1, l\rangle$ time and s_1 space, and let $A_2 \subset \mathbb{N}$ have a resolver R_{A_2} of t_2 time and s_2 space s.t. $t_2 \cdot \Delta \leq t_1$. Then $R_{A_1} \circ R_{A_2}$ resolves $A_1 \circ A_2$ in $\langle t_1 + \text{rate}(A_1) \cdot t_2, l + 1\rangle$ time and $s_1 + s_2$ space.

Proof: Let I be an interval, let $k = |I|$ and let T_I be the total time consumed by $R_{A_1} \circ R_{A_2}$ for resolving the integers (slots) of I. Then:

$$\begin{array}{rcll} T_I & \leq & t_1 \cdot (k + l) + t_2 \cdot |A_1 \cap I| & \text{since } R_{A_1}, R_{A_2} \text{ are of } \langle t_1, l\rangle,\ t_2 \text{ time, respectively} \\ & < & t_1 \cdot (k + l) + t_2 \cdot (\text{rate}(A_1) \cdot k + \Delta) & \text{since } A_1 \text{ is } \Delta\text{-smooth and } t_2 > 0 \\ & \leq & t_1 \cdot (k + l) + t_2 \cdot \text{rate}(A_1) \cdot k + t_1 & \text{since } t_2 \cdot \Delta \leq t_1 \\ & \leq & (t_1 + t_2 \cdot \text{rate}(A_1)) \cdot (k + l + 1) & \text{since } t_2, l \geq 0. \end{array}$$

That is, $R_{A_1} \circ R_{A_2}$ is indeed of $\langle t_1 + \text{rate}(A_1) \cdot t_2, l + 1\rangle$ time. □

For $A, D \subset \mathbb{N}$, D is a (ρ, Δ)-*quotient* of A if $D = A \circ B$ for some (ρ, Δ)-smooth B. A real function w is *steep* if $w : [0, 1] \to \mathbb{R}$ and $3\zeta(y - x) \leq w(x) - w(y)$ for any $0 \leq x \leq y \leq 1$. Such a function is monotonic decreasing and $w(y) + \zeta \cdot y \leq w(y \cdot 2/3)$. This and Lemmas 8 and 10 implies:

Lemma 11: Let w be a steep function with $w(1) \geq \Delta\zeta$ and let A be a Δ-smooth set having a resolver of $\langle w(\text{rate}(A)),\ l\rangle$ time and s space on a k-bit RAM. Let $r \leq 2/3$ be a k-bit rational and B be a $(r, 1)$-quotient of A. Then B has a resolver of $\langle w(\text{rate}(B)),\ l + 1\rangle$ time and $s + O(1)$ space on the k-bit RAM.

The following lemma shows that the time of a resolver can be 'deamortized'.

Lemma 12: Let $A \subset \mathbb{N}$ have a resolver of $\langle t, l\rangle$ time and s space on a certain RAM. Then A can be resolved in $O(t)$ time and $s + O(l)$ space on this RAM.

Proof: Let R be the above resolver of $\langle t, l\rangle$ time and s space, and consider the following resolver, R'. It follows the steps of R while spreading them evenly between its calls. To achieve that, R' *computes ahead* the output bits; that is, when it answers the i-th call it had already computed several additional bits (but keeps them a secret).

To this end, R' maintains a FIFO queue of pending output bits whose size is at most $l + 2$. In each call, R' simulates R and this simulation may generate several output bits which R' stores in the queue. The simulation terminates either when R' has simulated t steps of R or when the queue is full; R' then removes the first bit from the queue and returns this value. In the initial state the queue contains the first $l + 1$ bits of A. It remains to show that R' does not fail; that is, that the queue is never empty when R' tries to remove a bit from it.

To show that the i-th call to R' does not fail, let h be the maximum integer s.t. $h \leq i + 1$ and the queue contains $l + 1$ bits in the beginning of the h-th call. Such a h exists since the queue starts with $l + 1$ bits. If $h = i + 1$ then we are done. Otherwise, let $I' = [h, i]$; for any $j \in I'$, R' simulates exactly t steps of R during its j-th call. Let Y be the set of the steps of R simulated by R' during I'. Let I be the minimum interval of integers s.t. R performs all the steps of Y during its calls associated with the slots of this interval. Let x be the number of steps performed by R during I. We have:

$$|I'| \cdot t = |Y| \leq x \leq t \cdot (|I| + l)$$

The last inequality follows from the fact that R is of $\langle t, l\rangle$ amortized time. Thus, $|I'| \leq l + |I|$. During I' the resolver R' tries to remove exactly $|I'|$ bits from the queue and adds at least $|I| - 1$ new bits to the queue; since the queue starts with $l + 1$ bits, the i-th call to R' does not fail. □

4.2 The Flat Scheduling

This subsection presents the recursive technique in the context of the flat scheduling problem. The constructed schedule is 4-smooth and under the binary-fraction restriction it has $O(1)$ time resolvers. This subsection uses the following notations, some of which are already defined in the introduction. Let

Γ be a finite set of *clients*. A Γ-*allocation* is a system $\{A_\gamma \mid \gamma \in \Gamma\}$ of disjoint sets of natural numbers. Such an allocation is Δ-smooth if every A_γ is Δ-smooth; the allocation is *smooth* if it is Δ-smooth for some Δ. A *rate function* ρ *over* Γ is a function that assigns to each $\gamma \in \Gamma$ a real number $\rho(\gamma) \in [0, 1]$ called the *rate* of γ. For $D \subset \mathbb{N}$, a ρ-*allocation of* D is a system $\{A_\gamma \subset D \mid \gamma \in \Gamma\}$ of disjoint subsets of D with $\text{rate}(A_\gamma) = \rho(\gamma)$.

An *allocation request* is an ordered triple $\langle D, \Gamma, \rho\rangle$ s.t. Γ is a finite set of *clients*, ρ is a rate function over Γ, D is a subset of $\mathbb{N}$ having a rate and $\sum_{\gamma\in\Gamma} \rho(\gamma) \le \text{rate}(D)$. In the general case, a solution of such a request is a smooth ρ-allocation of D and a resolver of each of the sets of the allocation. However, we will usually have additional requirements on the request and on the solution; for example the requirement may be that the request obeys the 2/3-*portion restriction* — $\rho(\gamma) \le 2/3 \cdot \text{rate}(D)$ for each $\gamma \in \Gamma$. Under this formulation, the flat scheduling problem presented in the introduction is a special case of such a problem. An allocation request $\langle D, \Gamma, \rho\rangle$ is *tight* if $\sum_{\gamma\in\Gamma} \rho(\gamma) = \text{rate}(D) > 0$ and each $\rho(\gamma) > 0$.

The 'divide and conquer' idea, which is the essence of the recursive technique, is summarized by the following lemma which is based on composition of sets and follows directly from Lemma 2 and from the fact that $\text{rate}(A \circ B) = \text{rate}(A) \cdot \text{rate}(B)$ if the last two rates exists.

Lemma 13: Let $\langle D, \Gamma, \rho\rangle$ be a tight allocation request with $|\Gamma| > 1$. Then there are two tight allocation requests, $\langle D^1, \Gamma^1, \rho^1\rangle$ and $\langle D^2, \Gamma^2, \rho^2\rangle$, s.t. (D^1, D^2) is a partition of D, (Γ^1, Γ^2) is a partition of Γ and for each $j \in \{1, 2\}$:

a. The function ρ^j is the restriction of ρ to Γ^j.

b. The set D^j is a $(r, 1)$-quotient of D for some r.

c. Either $|\Gamma^j| = 1$ or $\text{rate}(D^j) \le 2/3 \cdot \text{rate}(D)$.

We refer to the pair of requests $\langle D^1, \Gamma^1, \rho^1\rangle$ and $\langle D^2, \Gamma^2, \rho^2\rangle$ provided by the above lemma as a 2/3-*decomposition of* $\langle D, \Gamma, \rho\rangle$. Note that if $\mathcal{A}^j = \left\{A_\gamma \mid \gamma \in \Gamma^j\right\}$ is a ρ^j-allocation of D^j for each $j \in \{1, 2\}$ then the *union of the allocations*, $\mathcal{A}^1 \cup \mathcal{A}^2 \triangleq \{A_\gamma \mid \gamma \in \Gamma\}$, is a ρ-allocation of D that conforms to the original request.

A *k-bit fraction* is a binary-fraction of the form $l/2^k$ for some $l \in \mathbb{Z}$; i.e., it is a number having a binary representation with k bits to the right of the binary point. Note that the set of the k-bit fractions is closed under addition and subtraction. Also, note that a k-bit fraction which is smaller than 1 is a $(k + 1)$-bit rational, and that the ratio of two, smaller than 1, k-bit fractions is a k-bit rational.

Lemma 14: Let $\Delta \ge 3$, let D be Δ-smooth and let $\langle D, \Gamma, \rho\rangle$ be an allocation request. Then there is a ρ-allocation of D $\{A_\gamma \subset D \mid \gamma \in \Gamma\}$ such that:

a. Each A_γ is $(\Delta + 1)$-smooth.

b. Let w be a steep function with $w(1) \ge \Delta\zeta$, let $\text{rate}(D)$ and all the $\rho(\gamma)$'s be k-bit fractions and let D have a resolver of $\langle w(\text{rate}(D)), l\rangle$ time and s space on the k-bit RAM. Then each A_γ has a resolver of $\langle w(\text{rate}(A_\gamma)) + \zeta \cdot \text{rate}(A_\gamma),\ l + |\Gamma|\rangle$ time and $s + O(|\Gamma|)$ space on the same RAM.

Proof: The proof is by induction on $|\Gamma|$. As usual, we prove a stronger and more complex version, which is derived from the original lemma as follows:

1. The condition that D is Δ-smooth is replaced with the condition that either D is Δ-smooth or $|\Gamma| = 1$ and D is $(\Delta + 1)$-smooth.

2. The request is assumed to be tight.

This variant clearly implies the lemma. As the case of $|\Gamma| = 1$ is trivial, let $|\Gamma| > 1$. Let $\langle D^1, \Gamma^1, \rho^1\rangle$ and $\langle D^2, \Gamma^2, \rho^2\rangle$ be a 2/3-decomposition of $\langle D, \Gamma, \rho\rangle$ provided by Lemma 13. By the induction hypothesis, the lemma holds for these two requests. The premise of the complex lemma (i.e., the first sentence of that lemma), for each $\langle D^j, \Gamma^j, \rho^j\rangle$, follows from Lemma 6, both when $|\Gamma^j| = 1$ and when $|\Gamma^j| > 1$. Let

$\{A_\gamma \mid \gamma \in \Gamma\}$ be the union of the two allocations provided by the lemma for these two requests. We now show that this allocation satisfies the conclusion of our lemma for the original request.

This allocation clearly satisfies statement (a). To establish statement (b), assume that the premise of statement (b) holds for $\langle D, \Gamma, \rho \rangle$, w, k and l. For any A_γ we now show that it satisfies the conclusion of statement (b). Let j be s.t. $\gamma \in \Gamma^j$.

Say first that $|\Gamma^j| = 1$. In this case $A_\gamma = D^j$ is a $(r, 1)$-quotient of D for some r. Since $\rho(\gamma)$ and rate(D) are k-bit fractions, r is a k-bit rational. By Lemmas 8 and 10, A_γ has a resolver of $\langle w(\text{rate}(D)) + \zeta \cdot \text{rate}(D),\ l+1 \rangle$ time and $s + O(1)$ space. The function $w(x) + \zeta \cdot x$ is monotonic decreasing; thus this resolver is of the required time.

Say next that $|\Gamma^j| > 1$. In this case D^j is a $(r, 1)$-quotient of D for some $r \leq 2/3$. Since all the given rates are k-bit fractions, r is a k-bit rational. Lemma 11 implies that the premise of statement (b) holds with D, Γ, ρ, w, k and l replaced by D^j, Γ^j, ρ^j, w, k and $l+1$. Thus, by the induction hypothesis, A_γ satisfies the conclusion of statement (b). □

In the context of the flat problem, the recursive technique is derived from the special case of Lemma 14 with $D = \mathbb{N}$, $\Delta = 3$ and, for example, $w(x) = 3\zeta(2-x)$. The resulting resolver is then 'deamortizied' by Lemma 12. This is summarized by the following theorem.

Theorem 2: Let $\{\rho_\gamma \mid \gamma \in \Gamma\}$ be an instance of the flat scheduling problem s.t. $\Sigma_{\gamma \in \Gamma}\ \rho_\gamma \leq 1$. Then there is a 4-smooth allocation $\{A_\gamma \mid \gamma \in \Gamma\}$ s.t. each A_γ has rate ρ_γ. Moreover, if all the ρ_γ are k-bit fractions then each A_γ has a resolver of $O(|\Gamma|)$ memory words and $O(1)$ time per slot on any $\Omega(k)$-bit RAM.

It is not hard to see that, under a certain encoding of certain subsets of $\mathbb{N}$, the time complexity of the allocation stage implicit in our construction is polynomial (in the length of a reasonable encoding of the allocation request). The resolver implicit in Lemma 14 is a recursive program composed of simple subroutines calling each other. However, the recursion in this program is actually a trivial left recursion that can be easily eliminated. This leads to a single program that resolves all the allocated sets; that is, the resolvers of distinct sets differ, not in the program, but in the width of the RAM, in the initial content of the memory and in the number of words used. It is not hard to see that the required memory initialization, and hence the entire resolver, can be constructed in polynomial time.

4.3 The Open-Market Scheduling

This subsection presents the recursive technique in the open-market context. Under the 2/3-portion restriction, the scheduling constructed here is 6-smooth and under the additional binary-fraction restriction it also has resolvers of $O(1)$ time and $O(n)$ space on a RAM of a small width.

Lemma 15: Let D be 6-smooth and let $\langle D, \Gamma, \rho \rangle$ be an allocation request obeying the 2/3-portion restriction. Then there is a ρ-allocation of D, $\{A_\gamma \subset D \mid \gamma \in \Gamma\}$, such that:

a. Each A_γ is 6-smooth.

b. Let w be a steep function with $2w(1) \geq 6\zeta$ and assume that rate(D) and all the $\rho(\gamma)$'s are k-bit fractions and that D has a resolver of $\langle 2w(\text{rate}(D)), l \rangle$ time and s space on the k-bit RAM. Then each A_γ has a resolver of $\langle 2w(\text{rate}(A_\gamma)),\ l + |\Gamma| \rangle$ time and $O(|\Gamma|) + s$ space on the same RAM.

Proof: Without loss of generality we assume that the allocation request is tight and that $|\Gamma| > 1$. Let $\langle D^1, \Gamma^1, \rho^1 \rangle$ and $\langle D^2, \Gamma^2, \rho^2 \rangle$ be a 2/3-decomposition of $\langle D, \Gamma, \rho \rangle$ provided by Lemma 13. Let $j \in \{1, 2\}$. By Lemma 13, D^j is a $(r, 1)$-quotient of D and the 2/3-portion restriction implies that $r \leq 2/3$. Thus, by Lemma 6(a), D^j is 5-smooth and $\langle D^j, \Gamma^j, \rho^j \rangle$ satisfies the premise of Lemma 14 for $\Delta = 5$. Let $\{A_\gamma \mid \gamma \in \Gamma\}$ be the union of the two allocations provided by Lemma 14 for the above two requests.

This allocation clearly satisfies statement (a). To establish statement (b), assume that the premise of this statement holds for the original items, D, Γ, ρ, k, w and l. For any A_γ we now show that it satisfies the conclusion of statement (b).

Let j be s.t. $\gamma \in \Gamma^j$. Define the function w' by: $w'(x) = w(x) + w(\text{rate}(D))$. The following requirments of Lemma 11 hold. The set D has a resolver of $\langle w'(\text{rate}(D)), l \rangle$ time and s space; the function w' is steep; $w'(1) \geq 6\zeta$; D is 6-smooth; $r \leq 2/3$ and is a k-bit rational; and the set D^j is a

$(r, 1)$-quotient of D. Thus, by Lemma 11, D^j has a resolver of $\langle w'(\text{rate}(D^j)), l+1\rangle$ time and $s + O(1)$ space.

Define the function w'' by: $w''(x) = 2w(x) - \zeta x$. We have $w'(\text{rate}(D^j)) = w(\text{rate}(D^j)) + w(\text{rate}(D)) \leq w(\text{rate}(D^j)) + w(\text{rate}(D^j)) - 3\zeta(\text{rate}(D) - \text{rate}(D^j)) \leq w''(\text{rate}(D^j))$. The following requirements of statement (b) of Lemma 14 hold. The function w'' is steep; $w''(1) \geq \Delta\zeta$, for $\Delta = 5$; and D^j has a resolver of $\langle w''(\text{rate}(D^j)), l+1\rangle$ time and $s + O(1)$ space. By this lemma, A_γ has a resolver of $\langle w''(\text{rate}(A_\gamma)) + \zeta \cdot \text{rate}(A_\gamma),\ l + 1 + |\Gamma^j|\rangle$ time and $s + O(|\Gamma^j|)$ space. By definition, $w''(\text{rate}(A_\gamma)) + \zeta \cdot \text{rate}(A_\gamma) = 2w(\text{rate}(A_\gamma))$; i.e., this resolver is of the required time. $\square$

The recursive technique in the open-market context is based on the above lemma. The set of slots of any dealer γ satisfies the following three invariants. It is 6-smooth, its rate is a binary-fraction, and it has a resolver of $\langle 2w(\rho(\gamma)),\ l\rangle$ time and $O(l)$ space for some l and for a ceratin steep w: for example, $w(x) = 3\zeta(2-x)$. Note that $\mathbb{N}$, the set of slots of the initial dealer, satisfies these invariants.

The following theorem summarizes the recursive technique in the open-market context. To this end, a dealer γ_1 is an *ancestor* of a dealer γ_2 if γ_2 bought his slots, directly or indirectly, from γ_1. A dealer γ_3 *affects* a dealer γ_2 if either γ_3 is an ancestor of γ_2 or γ_3 is a direct buyer from an ancestor of γ_2. For a dealer γ, let $\mathrm{F}(\gamma) \triangleq \{\gamma' \mid \gamma' \text{ affects } \gamma\}$. Note that the set of slots of a dealer γ depends only on the dealers of $\mathrm{F}(\gamma)$. The following theorem is implied by a simple induction using Lemma 15 augmented with Lemma 12.

Theorem 3: In the open-market context and under the 2/3-portion restriction the recursive technique allocates to any dealer (or client) γ a 6-smooth set with the appropriate rate. Moreover, if the rates of all the members of $\mathrm{F}(\gamma)$ are k-bit fractions then the set of γ can be resolved in $O(1)$ time and $O(|\mathrm{F}(\gamma)|)$ space on any $\Omega(k)$-bit RAM.

As in the flat scheduling problem, the allocation and resolvers provided by Theorem 3 can be constructed in polynomial time. We do not know if there is a scheduling technique that satisfies Theorem 3 without a big-portion restriction. However, a big-portion restriction is critical in Lemma 15, as implied by Lemma 7.

Tijdeman's Hierarchial Chairman Assignment Problem

Recall that Tijdeman's Hierarchial Chairman Assignment Problem [30] is defined as follows. There are several states, each having a positive weight and there is an hierarchial grouping of states into organizations. That is, states are grouped into unions, unions are grouped into federations, and so on. The overall organization has a yearly assigned chairman, and this assignment has to be smooth (and with the appropriate rate) w.r.t. any organization (state, union, federation, etc.). Our recursive scheduling technique, in the open-market context, solves a variant of this problem, as summarized by the following theorem.

Theorem 4: Any instance of the Hierarchial Chairman Assignment Problem obeying an α-portion restriction for $\alpha \in (0,1)$ has an allocation in which any organization is allocated an $O(1/(1-\alpha))$-smooth set with the appropriate rate. $\diamond$

This theorem is established as follows. In our setting, the dealers play the roles of the organizations and their rates is the accumulated rate of the corresponding states. Note that our scheduling problems do not require that all the slots are assigned to clients whereas such a requirement is mandatory in the chairman problem. However, it is easily verified that the recursive technique allocates all the slots when the rates sum to 1. The recursive technique is presented here with $\alpha = 2/3$ but it is not hard to generalize it to all $\alpha \in (0,1)$.

5 Imported Tools - Part 2

As said, this paper builds on the mathematical infrastructure of smooth sets developed in [27]. This section reviews notations and lemmas from there which are used by the intervals technique. Full proofs and elaborations are available in [27].

The intervals technique is based on a one-to-one partial function from the unit interval into the natural numbers s.t. an interval of length ρ is mapped onto a subset of the natural numbers with rate ρ. To formalized this, we henceforth extend any function f to be defined on any set Z (not necessarily a subset of the domain of f) by $f(Z) \triangleq \{f(z) \mid z \in Z \ \wedge \ f(z) \text{ is defined}\}$. For a real interval Z, let $\|Z\|$ denote the length of Z. A *shuffle* is a a partial one-to-one function f from the unit interval into the natural numbers s.t. for any interval $X \subset [0,1)$, $\text{rate}(f(X)) = \|X\|$. The intervals technique is based on a shuffle having an additional property that the image of any interval can be 'smoothed' into a smooth set by removing a small fraction of the interval, as stated by the next theorem which is the cornerstone of our intervals technique.

Theorem 5: [27] There is a shuffle f s.t. for any $0 < \epsilon$ and for any real interval $X \subset [0,1)$, there exists an interval $Y \subset X$ s.t. $\|X\|(1-\epsilon) \le \|Y\|$ and $f(Y)$ is $O(\log(1/\epsilon))$-smooth.

Note that the length of the above X intervals, as well as their endpoints, are not necessarily rational.

The shuffle satisfying Theorem 5, constructed in [27], is denoted μ and is called the *infinite bit reversal function*. This shuffle is related to the (finite) bit reversal permutation. The function μ is defined only on binary-fractions in the unit interval. (Recall that binary fractions are numbers of the form $l/2^j$ with $j, l \in \mathbb{N}$.)Let $x \in [0,1)$ be a binary-fraction and let $0.\alpha_0\alpha_1\alpha_2\cdots$ be its binary expansion. Then $\mu(x) \in \mathbb{N}$ is the number whose binary expansion is $\cdots\alpha_2\alpha_1\alpha_0$. This definition is meaningful since $\alpha_i \neq 0$ for finitely many i.

The intervals technique is based on the following two lemmas which use the following terminology. Let $n, j \in \mathbb{N}$. An *$(n \times j)$-interval* is an interval $X \subset [0,1)$ of the form $X = [l/2^j, (l+n)/2^j)$ for some $l \in \mathbb{N}$. We allow one or two of the parameters, n and j, to be replaced with a $*$ which stands for any natural number. For example, a *$(* \times j)$-interval* is an interval that is an $(n \times j)$-interval for some n (including the case of $n = 0$). An *extreme interval* is either the empty interval or a $(* \times *)$-interval X s.t. for some $(1 \times *)$-interval Y, $X \subset Y$ and X shares an endpoint with Y.

Lemma 16: [27] Let X be a non-empty $(n \times *)$-interval. Then:

a. The set $\mu(X)$ is $(\lceil \log n \rceil + 1)$-smooth.

b. Either X is an extreme interval or there are $\Delta_1, \Delta_2 \in \mathbb{R}$ and two disjoint extreme intervals X_1 and X_2 s.t. $X_1 \cup X_2 = X$, $\mu(X_1)$ is Δ_1-smooth, $\mu(X_2)$ is Δ_2-smooth and $\Delta_1 + \Delta_2 = (\lceil \log n \rceil + 1)$.

Lemma 17: [27] For any $0 < \epsilon$ and for any interval $X \subset [0,1)$ there is an $(n \times *)$-interval $Y \subset X$ s.t. $n < 4/\epsilon$ and $\|X\|(1-\epsilon) \le \|Y\|$.

6 The Intervals Technique

This section presents our second scheduling technique, the *intervals technique*. This technique allocates disjoint sets of slots by allocating disjoint real intervals and applying our shuffle function μ on these intervals. In order to avoid confusion between allocations of slots and allocations of intervals, we refer to the latter as *assigning*; namely, we allocate sets of slots and assign intervals of real numbers.

In the intervals technique (in both frameworks) the dealers/clients are highly insensitive to each other, as expressed by the following two properties which the recursive technique lacks. A client with a rational rate can resolve his set of slots in $O(1)$ time per slot using $O(1)$ memory words on a RAM whose width depends only on the rate of this (and no other) client. Namely, if his rate is a k-bit rational then any $\Omega(k)$-bit RAM will do. (Recall that in the recursive technique each resolver needs $O(n)$ space and the width of its RAM depends on the rates of many other clients/dealers.) The second property is that the intervals technique enables *piecewise allocation* of slots; that is, slots can be allocated to a client/dealer upon his arrival. This implies that in the flat scheduling framework the set allocated to a client depends only on the previous clients; in the open-market framework the set allocated to a dealer depends only on the set of his seller and on the rates of the previous buyers from this seller. Actually, this dependency on the sets of the previous clients/dealers is very restricted — only the sum of their rates (and not the manner this sum is divided into rates) is relevant. (In contrast, the recursive technique can allocate slots to a client/dealer only after the rates of all other clients/buyers from the relevant dealer are known).

In the flat framework the intervals technique produces an 11-smooth allocation. Unfortunately, this is under the restriction that the sum of the rates of the clients must not exceed 0.99. In other words, the technique utilizes only 99% of the resource. (Recall that the recursive technique utilizes all of the resource). The above numbers 0.99 and 11 are interdependent.

In the open-market framework the only restriction on the rates of the buyers is that their sum is not greater than the rate of the seller. Unfortunately, the set of slots allocated to a dealer is not necessarily smooth. (In the recursive technique these sets are smooth). However, any such set B has an 11-smooth subset with rate $0.99 \cdot \text{rate}(B)$. Thus, a dealer who intends to become a client with rate ρ should buy a set with rate $\rho' = (1/0.99) \cdot \rho$, and use a 0.99 fraction of this set. We refer to such ρ and ρ' as the *net rate* and the *gross rate* of the buyer. Dealers buy and sell sets having the appropriate gross rates while clients use sets having the appropriate net rate.

A natural extension of our scheduling problems concerns scheduling of several identical resources. A straightforward extension of the intervals technique works in this case of multiple resources and produces a scheduling having all the above desirable properties; this even under the restriction that a client may be served concurrently by at most one resource.

The performance of the intervals technique in the flat framework is not better than its performance in the more general open-market framework. Hence, we focus on the latter framework and the rest of this section is organized as follows. Subsection 6.1 establishes the allocation stage, Subsection 6.2 presents the resolution algorithm and Subsection 6.3 presents the extension of the technique to multiple resources.

6.1 The Allocation Process

The allocation performed by the intervals technique is based on Theorem 5 and Lemma 6(a) (but the fast resolutions of the allocated sets do not follow from Theorem 5, but from additional properties of our specific shuffle μ). The allocation process is composed of three stages. The first stage is applied to all dealers while the other stages are applied only to dealers turning into clients. These last stages are local ones, performed individually by each such dealer. The three stages are as follows:

The Dealers Stage: This stage assigns an interval X of length ρ' to any dealer with gross rate ρ'; the resulting set of slots, $\mu(X)$, has rate ρ' but is not necessarily smooth. All those that buy from a certain dealer receive disjoint sub-intervals of the interval of that dealer. Clearly, this stage can be performed in a piece-wise manner, and so that the interval assigned to a buyer depends only on the interval of its seller and on the sum of the rates of the previous buyers from this seller.

The objective of the last two stages is to 'smooth' and to adjust the rate of the allocated sets of slots. For these stages, consider a dealer with gross rate ρ' turning into a client with net rate $\rho = 0.99 \cdot \rho'$.

The Smoothing Stage: This stage replaces the interval X of length ρ' with an $(n \times j)$-interval Y s.t. $Y \subset X$, $\|Y\| \geq 0.99 \cdot \|X\|$, $n \leq 400$ and $j = \log(1/\rho) + O(1)$. This Y is provided by Lemma 17 for X and $\epsilon = 0.01$. Since $\rho \leq \|Y\| = n \cdot 2^{-j} < 400 \cdot 2^{-j}$, we have $j = \log(1/\rho) + O(1)$, as required. It follows from Lemma 16 and from the fact that $\lceil \log 400 \rceil = 9$ that the resulting $\mu(Y)$ is 10-smooth.

The Pruning Stage: This stage prunes, via set composition, the set $\mu(Y)$ into a $(\rho, 11)$-smooth set A. To that end, it picks a $(\rho/\|Y\|, 1)$-smooth set B (as provided by Lemma 2) and set $A = \mu(Y) \circ B$; by Lemma 6(a), A is $(\rho, 11)$-smooth.

6.2 The Resolution Algorithm

This subsection shows that each client has an efficient resolver of constant time and space. This resolver first resolves the set $\mu(Y)$ produced by the Smoothing Stage and then prunes this set.

We first show that $\mu(Y)$ can be resolved efficiently. To that end, for $x \in \mathbb{R}$ and $j \in \mathbb{N}$, let $\lfloor x \rfloor^j$ denote the maximal j-bit-fraction which is not larger than x. In other words, $\lfloor x \rfloor^j$ is the number obtained from the binary expansion of x by deleting all the bits rightwards to, and including, the $(j+1)$-th position after the point. For $j, n \in \mathbb{N}$, let $\nu^j(n) \triangleq \lfloor \mu^{-1}(n) \rfloor^j$. The following lemma is straightforward.

Lemma 18: For any $i \in \mathbb{N}$ and for any $(* \times j)$-interval Z: $i \in \mu(Z)$ iff $\nu^j(i) \in Z$.

Our fast resolver is based on Lemma 18 and on efficient generation of the sequence $\langle \nu^j(0), \nu^j(1), \ldots \rangle$. For this generation, we use a special type of counter, as follows. A *k-counter* is a data type with only one operation. This operation increases the value of the counter by 1, modulo k, and returns the new value. A (j,k)-*bit-reversal counter*, with $k \le 2^j$, is identical to the k-counter, except that the increment operation returns the *j-bit reversal* value of the counter; that is, if the j-bit binary expansion of the current value of the counter is $\alpha_{j-1}\cdots\alpha_0$ then the returned value is the binary number $\alpha_0\cdots\alpha_{j-1}$.

By definition, the value returned by the i-th call to the $(j,2^j)$-bit-reversal counter is $\nu^j(i)\cdot 2^j$. Hence, efficient implementation of this counter implies efficient generation of the sequence $\langle \nu^j(0), \nu^j(1), \ldots\rangle$.

Lemma 19: For any $k \le 2^j$, the (j,k)-bit-reversal counter can be implemented with $O(1)$ time and $O(1)$ space on any $\Omega(j)$-bit RAM.

Proof: Assume a $(j+1)$-bit RAM and that $k = 2^j$; the generalization of this case is straightforward. The state of the counter is maintained by two words: v, which contains the value of the counter, and r, which contains the j-bit reversal of v. Each increment updates the state of the counter as follows:

$$\begin{aligned}
&x \leftarrow \neg v \wedge (v+1)\\
&y \leftarrow 2^j/x\\
&v \leftarrow (v+1) \wedge (\neg 2^j)\\
&r \leftarrow r + y + \lfloor y/2 \rfloor - 2^j
\end{aligned}$$

In this program, 2^j and $\neg 2^j$ are constants and x and y are temporary variables. It is not hard to verify that this program indeed implements the $(j,2^j)$-bit-reversal counter. □

Thus we have:

Lemma 20: For any $j \in \mathbb{N}$, the infinite sequence $\langle \nu^j(0), \nu^j(1), \ldots\rangle$ can be generated in $O(1)$ space and $O(1)$ time per element on any $\Omega(j)$-bit RAM.

The former lemma implies the following one:

Lemma 21: For any $(* \times j)$-interval X, $\mu(X)$ can be resolved in $O(1)$ time and $O(1)$ space on any $\Omega(j)$-bit RAM.

Now we can summarize the resolution algorithm. Consider a client whose net rate $\rho \ne 0$ is a k-bit rational, and let Y be the $(* \times j)$-interval with $j = \log(1/\rho) + O(1)$ assigned to this client by the Smoothing Stage. As said, the resolution algorithm has two stages, as follows.

Resolving $\mu(Y)$: Since ρ is a k-bit rational and $\rho > 0$, we have $\rho \ge 1/2^k$. Thus, $j = \log(1/\rho)+O(1) \le \log(2^k) + O(1) = k + O(1)$. Hence, by Lemma 21, $\mu(Y)$ can be resolved in $O(1)$ time and $O(1)$ space on any $\Omega(k)$-bit RAM.

Pruning $\mu(Y)$: Let B be the 1-smooth set used by the Pruning Stage of the allocation and let $A = \mu(Y) \circ B$. By simple arithmetic, rate(B) is an $O(k)$-bit rational. Thus, by Lemma 8, B can be resolved in $O(1)$ time and $O(1)$ space on any $\Omega(k)$-bit RAM, and by Lemma 9, so can A.

This completes the resolution algorithm. The above construction leads to the following two theorems, one for each framework. The first theorem summarizes the intervals technique in the open-market context. (Recall that a dealer with gross rate ρ' has net rate $\rho = 0.99 \cdot \rho'$ and vice versa.)

Theorem 6: In the open-market context the intervals technique allocates to any dealer γ a set A_γ with the appropriate gross rate. When γ becomes a client he is allocated an 11-smooth set $A'_\gamma \subset A_\gamma$ with the appropriate net rate. Moreover, if the net rate of γ is a k-bit rational then A'_γ can be resolved in $O(1)$ memory words and $O(1)$ time per slot on any $\Omega(k)$-bit RAM.

Moreover, all the resolvers provided by Theorem 6 share the same program and can be generated in polynomial time.

The following theorem summarizes the intervals technique in the flat scheduling context; this theorem is a special case of Theorem 6, where all the clients buy directly from the initial dealer that owns the entire resource.

Theorem 7: Let $\{\rho_\gamma \mid \gamma \in \Gamma\}$ be an instance of the flat scheduling problem s.t. $\Sigma_{\gamma\in\Gamma}\, \rho_\gamma \le 0.99$. Then there is an 11-smooth ρ-allocation $\{A_\gamma \mid \gamma \in \Gamma\}$. Moreover, for each client γ, if ρ_γ is a k-bit rational then A_γ has a resolver of $O(1)$ memory words and $O(1)$ time per slot on any $\Omega(k)$-bit RAM.

6.3 Multiple Resources

A natural extension of our work concerns scheduling of m identical resources where the sum of the rates of the clients is (of course) at most m and the rate of each client is at most 1. If it is permitted to serve a single client by several resources in one slot then it is easy to transform any scheduling technique of a single resource into a scheduling technique of m identical resources , as follows. Split each slot into m consecutive sub-slots; schedule the resulting infinite sequence of sub-slots by the given technique on a single resource; replace the service provided by this resource during m consecutive sub-slots with the equivalence service provided by m resources in a single slot.

However, in some applications (e.g., CPU scheduling) a client may be served concurrently by at most one resource, and it was widely believed that this restriction "adds a surprising amount of difficulty" (to the scheduling problem) [28, 12]. In this subsection we show that our intervals technique has a straightforward extension to multiple resources, even under the restriction of no concurrent service. This extension has all the desired properties of the intervals technique. In particular, its resolvers are as efficient as those of the single resource case, and this even under the following additional burden on a resolver: in each time-slot the resolver tells, not only if the client is served in this time-slot, but also which resource serves it. Our extension has the additional interesting property that each client is served (not concurrently) by at most two resources.

Let a *service-slot* denote the atomic unit of service — servicing a single client by a single resource in a single time-slot. A system of m resources produces, in each time-slot, m service-slots, one by each resource. These service-slots are the entities that are allocated to the clients. Consider a system of m identical resources and let m be henceforth fixed. Formally, the service-slots are the members of the Cartesian product $\mathbb{N} \times \{0, \cdots, m-1\}$ where $\langle i, j \rangle$ denotes the service-slot produced by the $(j+1)$-th resource in the i-th time-slot.

In the following we adjust several notations of the context of a single resource to fit the context of multiple resources. Let Γ be a set of clients. A Γ-*allocation* is a system $\{S_\gamma \mid \gamma \in \Gamma\}$ of disjoint sets of service-slots. Clearly, we are interested in allocations having certain properties, as follows. A set S of service-slots is *non-concurrent* if any two of its members correspond to distinct time-slots. A Γ-allocation $\{S_\gamma \mid \gamma \in \Gamma\}$ is *non-concurrent* if each S_γ is non-concurrent. As said, we are interested in non-concurrent Γ-allocations.

For a set S of service-slots, let $\pi_1(S)$ denote the projection of S on its first coordinate — the natural numbers. Define $\mathrm{rate}(S) \triangleq \mathrm{rate}(\pi_1(S))$ if S is non-concurrent and otherwise $\mathrm{rate}(S)$ is undefined. A set S is Δ-smooth ((ρ, Δ)-smooth) if S is non-concurrent and $\pi_1(S)$ is Δ-smooth ((ρ, Δ)-smooth). Given a rate function ρ over Γ, a ρ-*allocation* is a Γ-allocation $\{S_\gamma \subset \mathbb{N} \mid \gamma \in \Gamma\}$ with $\mathrm{rate}(S_\gamma) = \rho_\gamma$. Such an allocation is Δ-smooth if every S_γ is Δ-smooth; the allocation is *smooth* if it is Δ-smooth for some Δ. A *resolver* of a non-concurrent set S of service-slots resolves in each time-slot i whether $i \in \pi_1(S)$, and if so computes the resource j s.t. $\langle i, j \rangle \in S$.

As in the single resource case, our extension applies a reduction from allocation of slots to allocation of intervals; however, it allocates sub-intervals of $[0, m)$ rather than of the unit interval. Thus, the extension is based on a one-to-one partial function $\hat{\mu}$ from the real interval $[0, m)$ onto the set $\mathbb{N} \times \{0, \ldots, m-1\}$ of service-slots s.t. an interval of length $\rho < 1$ is mapped onto a non-concurrent set of service-slots with rate ρ. Moreover, the interval $[i-1, i)$ is mapped onto the service-slots produced by the i-th resource. This partial function $\hat{\mu} : [0, m) \to \mathbb{N} \times \{0, \ldots, m-1\}$ is defined as follows:

$$\hat{\mu}(x) \triangleq \begin{cases} \text{undefined} & \mu(x \bmod 1) \text{ is undefined} \\ \langle \mu(x \bmod 1), \lfloor x \rfloor \rangle & \text{otherwise} \end{cases}$$

The following lemma follows directly from the definition of $\hat{\mu}$ and from the fact that μ is a shuffle; the specific definition of μ is irrelevant as far as this lemma is concerned. The lemma uses the following terminology: A real interval is *semi-open* if it is open on one side and closed on the other; i.e., it is of the form of $[x, y)$ or $(x, y]$ for some $x, y \in \mathbb{R}$.

Lemma 22: For any semi-open interval $X \subset [0, m)$ with $\|X\| \leq 1$, $\hat{\mu}(X)$ is non-concurrent and $\mathrm{rate}(\hat{\mu}(X)) = \|X\|$.

In this section we also adjust the notation of $(n \times j)$-intervals and its derivatives by replacing the requirement that these intervals are sub-intervals of the unit interval by the requirement that these intervals are sub-intervals of $[0, m)$. Many lemmas concerning these intervals also hold for the adjusted notations. In particular, we use the adjusted variants of the following two lemmas: Lemma 16 and Lemma 17 where the interval X is a sub-interval of $[0, m)$ whose length is at most 1 and at most 2, respectively.

The Allocation Process

Again, we present our extension in the open-market context. The allocation process in the case of multiple resources follows closely that of the single resource case. It has the same three stages augmented with an additional one. Again, clients perform all the stages while dealers perform only the first one.
The Dealers Stage: This stage assigns an interval X of length ρ' to any dealer with gross rate ρ'. These intervals are sub-intervals of $[0, m)$, rather than of $[0, 1)$ and their length may be greater than 1.
The Smoothing Stage: This stage replaces the interval X by an $(n \times j)$-interval Y s.t. $Y \subset X$, $\|Y\| \geq 0.99 \cdot \|X\|$, $n \leq 400$ and $j \leq \log(1/\rho) + O(1)$; such an interval Y exists by (the adjusted version of) Lemma 17. Note that the length of Y might be greater than 1 (but is less than 1/0.99).
The Chopping Stage: This is the additional stage which replaces Y by a sub-interval Y' having all the above properties of Y with the additional property that the length of Y' is at most 1. (The intervals Y and Y' may be identical.) By (the adjusted version of) Lemma 16, the resulting set of service-slots, $\hat{\mu}(Y')$, is 10-smooth.
The Pruning Stage: This stage prunes, via set composition, the set $\hat{\mu}(Y')$ into a $(\rho, 11)$-smooth set S of service-slots.

The Resolution Algorithm

Recall that a resolver of the above S needs to tell, for each time-slot, not only if the client is served in this time-slot but also which resource serves it. The resolution is similar to that of the single resource case and has the same two stages, as follows.
Resolving $\hat{\mu}(Y')$: The above $(n \times j)$-interval Y' is of the form $(Z_1 + l) \cup (Z_2 + l + 1)$ where $l \in \mathbb{N}$ and Z_1 and Z_2 are disjoint $(* \times j)$-intervals which are sub-intervals of $[0, 1)$. Lemma 21 and the definition of $\hat{\mu}$ imply that $\hat{\mu}(Y')$ can be resolved (in the stronger sense) in $O(1)$ time and $O(1)$ space on any $\Omega(j)$-bit RAM. If the net rate ρ is a k bit rational, then $j = k + O(1)$; thus, a $\Omega(k)$-bit RAM is applicable.
Pruning $\hat{\mu}(Y')$: The above $\hat{\mu}(Y')$ is pruned, by set composition, into the set S, as in the single resource case.

The following Theorem summarizes the extension of the intervals technique to multiple resources in the open-market context.

Theorem 8: Consider a system of m identical resources. In the open-market context our extension assigns to any dealer γ a real interval $X_\gamma \subset [0, m)$ whose length is the gross rate of γ. When γ becomes a client he is allocated a (non-concurrent) 11-smooth set of service-slots, S_γ, with the appropriate net rate. Moreover, if the net rate of γ is a k-bit rational then S_γ can be resolved in $O(1)$ time and $O(1)$ space on any $\Omega(k)$-bit RAM.

The following theorem summarizes our extension in the flat scheduling context and is a special case of Theorem 8, as in the single resource case.

Theorem 9: Consider a system of m identical resources and let $\{\rho_\gamma \mid \gamma \in \Gamma\}$ be an instance of the flat allocation problem s.t. $\sum_{\gamma \in \Gamma} \rho_\gamma \leq 0.99 \cdot m$ and each $\rho_\gamma \leq 1$. Then there is a (non-concurrent) 11-smooth ρ-allocation $\{S_\gamma \mid \gamma \in \Gamma\}$. Moreover, for each client γ, if ρ_γ is a k-bit rational then S_γ can be resolved in $O(1)$ time and $O(1)$ space on any $\Omega(k)$-bit RAM.

7 An Application

Here is an application that illustrates the advantages of the open-market framework and the distributed and predictable nature of our approach. They enable cascading of several virtual communication links into a single virtual link in a distributed open-market manner, as follows. Consider two cables, one from London to New York and the other from New York to Tokyo. Assume that these cables are multiplexed at the packet level in the same manner as our slot oriented resource and assume that the duration of a time-slot, as well as the volume transmitted in each time-slot, is identical in both cables. There is no buffering associated with the cables; i.e., a client is responsible to deliver his packets to one end of the cable at the appropriate slots and to receive them immediately when they emerge at the other end.

An entrepreneur dealer who buys, in our open-market framework, two (ρ, Δ)-smooth sets of slots, one of each cable, can create a virtual link from London to Tokyo by installing a device in New York that forwards his packet-stream from the first cable to the appropriate slots of the second cable. It is not hard to see [26] that this device needs to store less than 2Δ packets and that the delay of this newly created virtual link is less than $2\Delta/\rho + D$, when D is the sum of the delays of the two physical cables. The delay of the virtual link should be constant; to this end, the dealer installs another device in Tokyo which delays the packets so that they all have the same accumulated delay, equal to the above bound. This device needs no time-stamp and works as follows. Let A be the set of slots of the first cable owned by our dealer; the device resolves the slots of A delayed appropriately and releases packets only on those slots. It is not hard to see [26] that this device needs to store at most 3Δ packets. Our dealer can now sell fractions of this new London-Tokyo virtual link to dealers or clients. The fact that this virtual link passes through two physical cables is transparent to the buyers.

As mentioned in Section 2, the above is an example of an application for which the framework of the periodic scheduling problem is unsuitable. Recall that under this framework there is a set of n tasks (clients) and each task γ is associated with two natural numbers $e_\gamma \leq p_\gamma$. The schedule has to allocate the resource to a task γ for exactly e_γ time-slots in each interval $[p_\gamma k, p_\gamma(k+1))$ for all $k \in \mathbb{N}$. Consider the above application, and assume that our entrepreneur dealer buys a fraction of $(j+1)/(2j)$ of each cable, where j is an arbitrary large number, and those fractions are scheduled as per the periodic scheduling problem. It is possible that in an interval $[2j \cdot k, 2j \cdot (k+1))$, our dealer receives the first $j+1$ slots of the London-New York cable and the last $j+1$ slots of the New York-Tokyo cable. In this case, each of the devices needs to store $\Omega(j)$ packets and the delay of the virtual link is also $\Omega(j)$. As described above, under a smooth schedule these numbers are bounded by small constants which are independent of j.

Acknowledgments

We wish to thank Martin Cohn, Guy even, Shlomo Moran and Adi Rosen for helpful discussion and references. We wish to thank Arie Freund for reviewing an early draft of this paper and providing us with many suggestions that improve the organization of this paper. We are deeply grateful to the late Professor Shimon Even for his inspiration and encouragement.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: data management for asymmetric communication environments. In *Proceedings of ACM SIGMOD Conference*, pages 199–210, 1995.

[2] M. Adler, P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, and M. Paterson. A proportionate fair scheduling rule with good worst-case performance. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 101–108, 2003.

[3] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 35–43, 2000.

[4] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proc. of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306, 2000.

[5] J. H. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Computer and System Sciences*, 68:157–204, 2004.

[6] B. Andersson, S. K. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. Technical Report TR01-016, Department of Computer Science, University of North Carolina - Chapel Hill, 2001.

[7] S. Anily, C.A. Glass, and R. Hassin. The scheduling of maintenance service. *Discrete Applied Mathematics*, 82:27–42, 1998.

[8] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling (extended abstract). In *the Ninth Symposium on Discrete Algorithms*, pages 11–20, 1998.

[9] A. Bar-Noy, V. Dreizin, and B. Patt-Shamir. Efficent periodic scheduling by trees. In *the 21st INFOCOM*, pages 791–800, 2002.

[10] A. Bar-Noy, A. Mayer, B. Schieber, and M. Sudan. Guaranteeing fair service to persistent dependent tasks. *SIAM J. Of Computing*, 27(4):1168–1189, 1998.

[11] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir. Nearly optimal perfectly-periodic schedules. In *the 20th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 107–116, 2001.

[12] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algoritmica*, 15(6):600–625, 1996. (Extended abstract was presented at The 25th Annual ACM Symposium on the Theory of Computing. May 1993.).

[13] S. K. Baruah, J. Gehrke, and G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 280–288. IEEE Computer Society Press, 1995.

[14] S. K. Baruah, J. Gehrke, G. Plaxton, I. Stoica, H. Abdel-Wahab, and K. Jeffay. Fair on-line scheduling of a dynamic set of tasks on a single resource. *Information Processing Letters*, 64(1):43–51, 1997.

[15] S. K. Baruah, R. R. Howell, and L. E. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[16] J. C. R. Bennet and H. Zhang. WF^{2}Q: Worst-case Fair Queueing. In *the Fifteenth INFOCOM*, pages 120–128. IEEE, 1996.

[17] Z. Brakerski, A. Nisgav, and B. Patt-Shamir. General perfectly periodic scheduling. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, pages 163–172, 2002.

[18] A. Chandra, M. Adler, P. Goyal, and P. Shenoy. Surplus fair scheduling: A Proportional-Share CPU scheduling algorithm for symmetric multiprocessors. In *Proceedings of the USENIX 4th Symposium on Operating System Design and Implementation*, pages 45–58, 2000.

[19] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, pages 3–14, 2001.

[20] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press and McGraw-Hill Book Company, second edition, 2001.

[21] P. Holman and J. Anderson. Guaranteeing pfair supertasks by reweighting. In *Proceedings of the 22nd IEEE Real-time Systems Symposium*, pages 203–212, 2001.

[22] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12st IEEE Real-time Systems Symposium*, pages 129–139, 1991.

[23] C. Kenyon, N. Schabanel, and N. Young. Polynomial-time approximation scheme for data broadcast. In *Proceeding of the 32rd ACM Symposium on the Theory of Computing*, pages 659–666, 2000.

[24] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and avarage case behaivior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[25] A. J. Lincoln, S. Even, and M. Cohn. Smooth pulse sequences. In *Proceedings of the Third Annual Princeton Conference on Information Sciences and Systems*, pages 350–354, 1969.

[26] A. Litman and S. Moran-Schein. On centralized smooth scheduling. Technical Report CS-2005-04, Department of Computer Science, Technion - Israel Institute of Technology, 2005. Available at: www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi?2005/CS/CS-2005-04.

[27] A. Litman and S. Moran-Schein. On smooth sets of integers. Technical Report CS-2005-02, Department of Computer Science, Technion - Israel Institute of Technology, 2005. Available at: www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi?2005/CS/CS-2005-02.

[28] C. L. Liu. Scheduling algorithms for multiprocessors in hard-real-time enviroment. *JPL space program summary* 37-60, vol. II, Propulsion Lab., Calif. Inst. of Tech., Pasadena, CA, pages 28–37, 1969.

[29] M. Moir and S. Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *The 20th IEEE Real-Time Systems Symposium*, pages 294–303, 1999.

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