Adaptive General Perfectly Periodic Scheduling

Shailesh Patil and Vijay K. Garg
{patil, garg}@ece.utexas.edu
Department of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX 78712

***Abstract*— We propose an adaptive algorithm *Adaptmin* to create perfectly periodic schedules. A perfectly periodic schedule schedules a client regularly after a predefined amount of time known as the period of the client. The periodicity of such schedules can be used to save battery life of nodes in a wireless network. The quality of a perfectly periodic schedule is a function of the ratio between the granted and requested periods. We find a worst case performance bound on the quality of schedules produced by *Adaptmin*. We also deduce family of input instances where either *Adaptmin* does no worse than previous work, or always outperforms previous work. The better performance of *Adaptmin* is confirmed by simulations results for randomly generated input instances. The simulation results also show that the schedules produced by *Adaptmin* can be more than 25% efficient. We also propose a variant of *Adaptmin* which is computationally much less demanding compared to previous work, but is very close to *Adaptmin* in terms of efficiency. Finally we compare our algorithms to optimal scheduling, simulation results indicate that our algorithms performance is close to optimal scheduling.**

Keywords: Scheduling, Distributed systems, Algorithms, Analysis of algorithms

I. Introduction

Power consumption is one of the major challenges faced in the design of portable wireless devices. In an ad-hoc network, multiple portable devices may communicate with each other for different services. This requires the devices to be awake throughout the session of interaction with another device. One way to reduce power consumption is to schedule devices after a fixed period of time, so that they are awake only when they are being served.

Consider a scenario where a device has to serve requests from multiple devices. In order to do so, a schedule has to be defined that grants access to every requesting client device. In our setup, for any client i, a request is defined by a tuple (b_i, τ_i), where b_i is the requested length and τ_i is the requested time period. In other words, client i requests to be served for b_i consecutive time slots every τ_i time slots. Based on the requests, the serving device forms a schedule S. A schedule is said to be *perfectly periodic* [8] if each client is scheduled *exactly* every τ_i^S time slots, where τ_i^S is called the period of client i for schedule S. Note that τ_i^S may be different from the requested time period τ_i. In fact for some set of requests, it is impossible to give the exact requested period to all the clients. If $\tau_i^S = \tau_i$ for all i, then S is called a distortion free schedule. The efficiency of a schedule is a function of the ratio of τ_i^S and τ_i. In this paper we propose an adaptive algorithm *Adaptmin* that produces efficient perfectly periodic schedules in polynomial time.

To formally define the efficiency of a schedule, we introduce some notation. The set of requests or jobs is known as an'instance' and is denoted by $J = \{(b_i, \tau_i)_{i=1}^n\}$, where n denotes the number of jobs. The requested bandwidth of job i is defined as $\beta_i = \frac{b_i}{\tau_i}$. The total bandwidth of an instance J is defined to be $\beta_J = \sum_{i=1}^n \beta_i$. To evaluate the quality of a perfectly periodic schedule, two measures are suggested in [7] [8]. The first measure is an average measure known as $C_{AVE}(J, S)$, and is defined as $C_{AVE}(J, S) = \frac{1}{\beta_J} \sum_{i=1}^n \beta_i \frac{\tau_i^S}{\tau_i}$. The second measure is a maximum measure and is defined as $C_{MAX}(J, S) = \max\{\frac{\tau_i^S}{\tau_i} | i \in J\}$.

The original motivation for studying perfectly periodic schedules was broadcast disks. In a broadcast disk system a server broadcasts data "pages" to clients in a perfectly periodic manner; allowing clients to sleep until its desired page is broadcast. Note that this is a special case of the portable wireless device scenario where two-way communication is required. Other motivations include teletext systems [1] [2], chairperson assignment [11], machine maintenance [12] [3] [4] and fair time scheduling problems.

It has been shown in [5] by Bar-Noy et. al., that even deciding whether a given set of requests can be scheduled in a perfectly periodic manner is NP hard. However in [7] [6], polynomial complexity tree based scheduling algorithms have been presented as a suboptimal solution to the problem for the case where $\forall i \in J$, $b_i = 1$. This work has been extended in [8] to requests with b_i greater than or equal to one. To the best of our knowledge, this is the only proposed solution for the general case. Tree-based perfectly periodic scheduling has also been investigated for possible inclusion into the IEEE's 802.11e WAN QoS standard. Dhanakoti et. al. in [10] propose a binary-tree-based perfectly periodic scheduling scheme for fault avoidance in 802.11 WLANs. In this paper we study the case where b_i is greater than one and compare our algorithm to that proposed in [8].

This paper is divided into five sections. Section II presents an overview of the algorithm presented in [8]. This is followed by a description of the proposed algorithm *Adaptmin*. Section III of the paper gives analytical bounds for the same. We also present families of instances where *Adaptmin* either does no worse than previous work or always outperforms it. A computationally light variant of *Adaptmin*, *Adapt* and simulation results are presented in section IV. Section V concludes the paper.

Algorithm **Adaptmin**

Input: Instance *J*

Output: Schedule *S*

Steps:

1. Round the periods to nearest power of 2, i.e. $\tau_i' = 2^{\lceil \log \tau_i \rceil}$ for all *i*.
2. Sort the jobs in the increasing order of periods.
3. Find the minimum period $t' = min\{\tau_i' \mid i \in J\}$, maximum period $T' = max\{\tau_i' \mid i \in J\}$ and the number of intervals $\alpha = T'/t'$.
4. For each job *i*, starting with the first job, follow steps 5 to 7.
5. Try to schedule job *i* in the interval with least job length scheduled among the first $\alpha_i = \tau_i'/t'$ intervals.
6. If the job is scheduled, then schedule the job in all intervals that are at a multiple of α_i intervals away (from the first interval in which the job is scheduled) and go back to step 4. Else, go to the next step.
7. Find the interval within the first α_i intervals that requires the minimum expansion to accommodate job *i*, say interval *k*. Expand all intervals by the minimum expansion. Schedule job *i* in interval *k* and the intervals that are at a multiple of α_i intervals away from *k*. Go to step 4.

Fig. 1. Algorithm for *Adaptmin*

II. *Adaptmin*

We define some notation that will be needed during the description and analysis of proposed algorithms. The free bandwidth of an instance is defined as $\Delta_J = 1 - \beta_J$. The extent of a instance is defined as $R_J = \frac{\max\{b_i | i \in J\}}{\min\{\tau_i | i \in J\}}$. The minimum requested period of an instance is denoted by $t_J = \min\{\tau_i | i \in J\}$, while the maximum period is denoted by $T_J = \max\{\tau_i | i \in J\}$. We will drop the subscript J when there is no ambiguity about the instance being referred to.

As mentioned earlier, an algorithm called *A* was proposed in [8] to create perfectly periodic schedules. It was shown that if all the periods of the instance J are of the form $\tau_i = 2^{m_i} e$, where e is a constant, and $\Delta \geq R$, then a distortion free perfectly periodic schedule exists and can be created using the proposed tree based algorithm. It was further shown that for any instance with periods that are of the form $2^{m_i} e$, if all the periods are scaled by a factor of $\beta + R$, then for the resulting instance $\Delta \geq R$. Algorithm *A* first rounded the periods to powers of two. Then, all the periods were scaled by a factor of $\beta' + R'$, where β' and R' are the bandwidth and extent of the input instance *after* rounding periods to the nearestpower of two. A perfectly periodic schedule was produced using tree scheduling for the scaled up instance. However, the requirement $\Delta \geq R$ is very restrictive. Consider the instance $J = \{(1,1)\}$. In this example, $\Delta = 0$ and $R = 1$, but it is easy to produce a distortion free schedule for the instance. Algorithm *A* will produce a schedule that doubles the period. A more complex example is when $J = \{(2,4),(1,8),(2,8),(1,16)\}$. A distortion free schedule can also be formed for this instance, but *A* expands each period by a factor of $\frac{23}{16}$. One can easily come up with more examples where $\Delta < R$, but a distortion free perfectly periodic schedule exists.

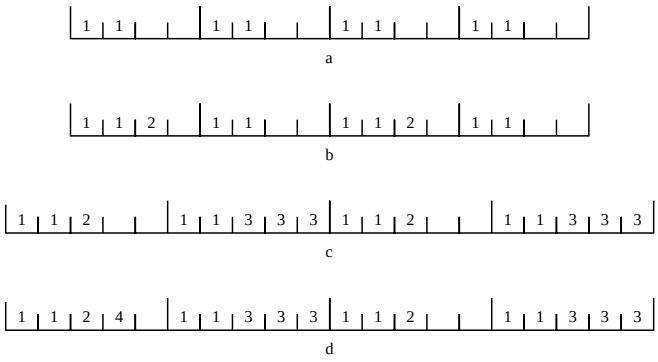

Fig. 2. *Adaptmin* scheduling instance $J = \{(2,3),(1,5),(1,13),(3,8)\}$

Fig. 1 gives the algorithm for *Adaptmin*. We explain this proposed algorithm with an example. Consider an instance $J = \{(2,3),(1,5),(1,13),(3,8)\}$. The first step is to round all periods up to nearest power of two, i.e., $\tau_i' = 2^{\lceil \log_2 \tau_i \rceil}$. This gives us the modified instance $J' = \{(2,4),(1,8),(1,16),(3,8)\}$. Step 2 requires that the input instance be sorted in the ascending order according to periods, i.e., $\forall i,j \in J$, if $\tau_i \leq \tau_j$, then $i \leq j$. No order is required among jobs with equal periods. Note that algorithm *A* also requires the same kind of ordering in the input instance. The sorted instance is given by $J' = \{(2,4),(1,8),(3,8),(1,16)\}$.

Like algorithm *A*, our algorithm also equally scales the rounded periods τ_i', i.e., $\frac{\tau_i^S}{\tau_i'}$ will be equal for all i. This is desirable, since it allows the maximum change in the period of a particular job to be bounded and thus ensures some level of fairness among the jobs. Also, scaling rounded periods differently may lead to higher computational complexity.

Continuing with the example, we describe step 3 now. Step 3 calculates the minimum rounded period $t' = 4$, the maximum rounded period $T' = 16$. The algorithm divides the total schedule into $\alpha = T'/t'$ intervals, each interval of length t'. For the given instance, the number of intervals is $\alpha = 4$, with each interval of length $t' = 4$. We meet the perfectly periodic property of the schedule being produced by ensuring that for any job i with requested period τ_i', the job is scheduled within $\alpha_i = \tau_i'/t'$ intervals.

In step 4, we consider the first job $(2,4)$. Step 5 tries to schedule this job in the first $\alpha_1 = 1$ interval(s). Since there is space, it is possible to do so. Step 6 then schedules the job in all the intervals that are $\alpha_1 = 1$ interval away. This is shown in Fig. 2(a), where 1 denotes the first job. Next, we loop back to step 4 and consider the second job $(1,8)$. Here $\alpha_2 = 2$, and since both the first and second interval have equal empty space, step 5 schedules the job in the first interval, while step

6, schedules it in the third. Fig. 2(b) illustrates this. We again loop back to step 4 and now consider the third job, i.e., $(3, 8)$. Since α_3 is also equal to 2, this job also has to be scheduled in the first two intervals. However in step 5, we are unable to find enough space in either of the first two intervals. This means that there is a need to expand the intervals, for this we proceed to step 7. Step 7 finds the interval among the first two intervals that requires the minimum expansion to accommodate $(3, 8)$. This is clearly the second interval, and it requires an expansion of one slot. All intervals are expanded by a unit slot, and $(3, 8)$ is scheduled in the second and the fourth interval, as shown in Fig. 2(c). We again loop back to step 4 and consider the last job $(1, 16)$. Step 4 places the job in the first interval, and this completes the schedule. The completed schedule is shown in Fig. 2(d). This completes our description of the algorithm. One can think of *Adaptmin* as a 'best fit' solution, because it greedily tries to find the best interval to schedule a job.

III. Analysis of *Adaptmin*

A. Performance bound

In this section we analyze the performance of *Adaptmin* by finding the upper bounds on C_{AVE} and C_{MAX} of schedules produced by the algorithm. We first consider the case where all the periods are powers of two, i.e., of the form $\tau_i = 2^{m_i}e$, where e is a constant. In this case, all time periods are of the form $2^k t$, where k is an integer that varies from 0 to $K := \log_2(T/t)$. We group jobs of equal period together. Let $Q_k = \{(b_i, \tau_i) | \tau_i = 2^k t \ \&i \in J\}$ denote the set of jobs with periods equal to $2^k t$. The sum of lengths for jobs with equal periods is denoted as $B_k = \sum_{i \in Q_k} b_i$. Define

$$L_k(b) = \max_{q \in P(Q_k)} \{\sum_{i \in q} b_i \mid \sum_{i \in q} b_i \geq b \text{ s.t. } \forall j \in q \sum_{i \in q-j} b_i < b\},$$

where $P(Q_k)$ is the power set of Q_k. Let use describe $L_k(b)$ in a more verbose manner. To find the quantity, we first find all the subsets q of Q_k, such that $\sum_{i \in q} b_i \geq b$ and if any job j is removed from q, then $\sum_{i \in q-j} b_i < b$. Then among such subsets, we choose the set for which the sum of job lengths is maximum, $L_k(b)$ denotes that maximum.

We define one more quantity before presenting the bound,

$$J_\rho = \{(b_i, \tau_i) \in J | \log_2(\frac{\tau_i}{t_J}) \leq \rho\},$$

i.e., an instance containing a subset of jobs of J with periods less than or equal to $2^\rho t_J$.

Lemma 3.1: For an instance J with periods of the form $\tau_i = 2^{m_i}e$, the maximum length in an interval for a schedule S created using *Adaptmin* is at most

$$\sum_{k=0}^{K} L_k(B_k/2^k),$$

Proof: The proof is shown by performing induction on $\bar{J}_\rho := J_\rho \cup \{(0, T_J)\}$. The bound is obvious on a schedule created for $\bar{J}_0$. Assume the bound is true on a schedule created for $\bar{J}_{g-1}$.

To obtain a schedule for $\bar{J}_g$ one needs to add jobs with period $2^g t_J$ to the schedule created for $\bar{J}_{g-1}$. Let U be the set of intervals and l_u^{g-1} be the length scheduled in interval u, for $\bar{J}_{g-1}$. By assumption

$$l_u^{g-1} \leq \sum_{k=0}^{g-1} L_k(B_k/2^k)$$

($B_K = 0$ in $\bar{J}_{g-1}$).

Considering the worst case, we assume

$$\forall u \in U, \ l_u^{g-1} = \sum_{k=0}^{g-1} L_k(B_k/2^k).$$

Jobs of period $2^g t_J$ are divided among 2^g intervals. Consider the case where *not* all jobs of period $2^g t_J$ have been scheduled, let $l(u)$ be the length scheduled in interval u until now. Then if

$$\exists \ u \in U, \text{ s.t. } l(u) - l_u^{g-1} \geq B_g/2^g,$$

by averaging argument,

$$\exists \ r \in U, \text{ s.t. } l(r) - l_r^{g-1} < B_g/2^g.$$

Since *Adaptmin* schedules job in the interval with the most empty space, none of the remaining jobs will be scheduled in u. Then, the maximum length added to any interval will be $L_g(B_g/2^g)$. So,

$$l_u^g \leq \sum_{k=0}^{g} L_k(B_k/2^k).$$

This completes the proof. ■

Theorem 3.2: For an instance J with periods of the form $\tau_i = 2^{m_i}e$ and $\sum_{k=0}^{K} L_k(B_k/2^k) > t$, the $C_{AVE}(J, S)$ for a schedule S created using *Adaptmin* is less than

$$C_b = \frac{\sum_{k=0}^{K} L_k(B_k/2^k)}{t}$$

Proof: The proof follows easily from Lemma 3.1. ■

Note that when $\sum_k L_k(B_k/k) \leq t$, $C_{AVE} = 1$.

Consider an instance $J = \{(2, 4), (1, 8), (1, 8), (2, 16)\}$, a schedule created using *Adaptmin* for J trivially achieves the bound. This shows that the bound is tight. For instances with jobs having periods that are not power of two, the bound can be trivially shown to be $2C_b$.

B. Comparison with A

Since both *A* and *Adaptmin* round periods to nearest power of two, it is sufficient to compare the distortion produced by each after rounding. Recall that J' denotes the modified instance J after rounding of periods, with the rounded off periods being denoted by τ_i'. The minimum and maximum rounded periods denoted by t' and T' respectively, while the bandwidth and extent for J' are denoted by β' and R'.

Note that $\sum_{k=0}^{K} B_k/2^k t = \beta'$, so if $(C_b - \beta') \leq R'$, then *Adaptmin* will *always* produce a more efficient schedule than that produced by algorithm *A*. However, if $(C_b - \beta') > R'$,

then *Adaptmin may* still outperform *A*, since C_b may not be achieved while *A* always scales periods by $\beta' + R'$. This indicates that in general *Adaptmin* may outperform *A*.

From the above discussion, one can come up with family of instances where *Adaptmin* will always outperform *A*. Examples of such family of instances are instances that *after* rounding of periods contain jobs with at most two different periods.

Consider the case where J' contains jobs with exactly one period. Then *Adaptmin* will expand rounded periods by a factor of $\max\{1, \beta'\}$. In other words *Adaptmin* will produce no distortion after rounding for $\beta' < 1$. While *A* will expand the round periods by a factor of $\max\{1, \beta' + R'\}$ (depending on whether $\Delta' \geq R'$, or $1 \geq \beta' + R'$). Then it is clear that *Adaptmin* will always outperform *A*, i.e., i.e., not only *Adaptmin* will produce distortion for a subset of instance for which *A* produces distortion, but the distortion produced by *Adaptmin* will be less.

Now consider the case where J' contains jobs with exactly two different periods. If $\Delta' < R'$, *Adaptmin* will expand rounded periods *at most* by a factor of $\beta' + R'$, while *A* will *always* expand rounded periods by a factor of $\beta' + R'$. However if $\Delta' \geq R'$, both the algorithms do not expand the rounded periods. It is clear that for such a family of instances *Adaptmin* will never do worse than *A*. An example instance that contains exactly two different periods after rounding is $J = \{(1,3),(1,4),(2,5),(1,7),(1,8)\}$, so a fairly large number of instances may belong to the family.

Let us describe another family where *Adaptmin* always outperforms *A*. Define c to be the maximum job length in an instance, i.e., $c := \max\{b_i | i \in J\}$. Note that $c = Rt = R't'$. Consider a family of instances where each job after rounding contains different periods, i.e., every job in J' has a unique period after rounding. We claim that for such a family of instance *Adaptmin* always outperforms *A*. To validate our claim, we present the following lemma.

Lemma 3.3: Consider an instance J such that the rounded period instance J' contains jobs with unique periods. Then the maximum length scheduled in an interval of a schedule created using *Adaptmin* for the J is bounded by $B_0 + c$, where B_0 is the length of the job with minimum period t in J.

Proof: Recall that the first step of *Adaptmin* is to round the job periods to their nearest power of two. This is followed by sorting the jobs in increasing order of their periods.

We give the proof by induction on jobs being scheduled. It is clear that the bound will hold when the first job is scheduled. By assumption, the bound holds after $i-1$ jobs have been scheduled. We need to prove that the bound holds after scheduling of job i.

Let $\tau_i' = 2^g t'$, then job i has to be scheduled once within the first 2^g intervals. Consider jobs $2, \ldots, i-1$, i.e., all the jobs scheduled before job i excepting the first one. All of them have unique periods of the form 2^k, where $k < g$. Then each job will be scheduled in 2^{g-k} intervals. Note that $\forall g$,

$$\sum_{k=2}^{g-1} 2^{g-k} < 2^g.$$

Then there exists at least one interval where the scheduled length is only B_0, the lengths scheduled in all other intervals will be equal or greater. So the maximum length scheduled in an interval after job i has been scheduled is $B_0 + c$. This completes the proof. ■

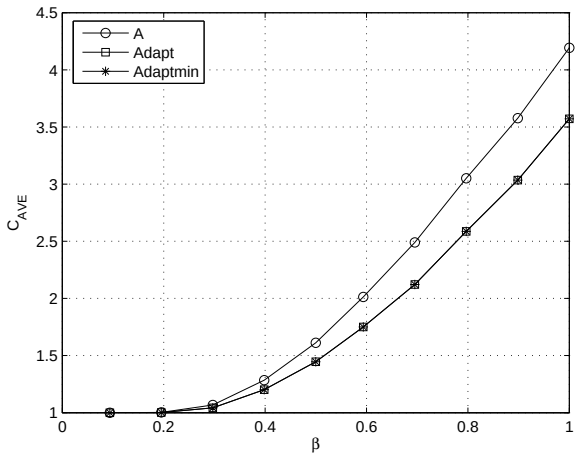

Fig. 3. Performance of *A*, *Adapt* and *Adaptmin*

When the number of jobs n is greater than 1, $(B_0/t') < \beta'$ it is clear that for instances which contain unique periods after rounding *Adaptmin* will always outperform *A*. The case where $n = 1$ has been discussed earlier. One can extend our claim to the case where J' contains job with unique periods for periods greater than t'.

We focus on the maximum measure now. It is quite easy to see that the maximum measure $C_{MAX}(J, S)$ for *Adaptmin* is bounded by $2C_b$, while for *A* the bounded is $2(\beta + R)$. If $C_b \leq (\beta + R)$, then clearly *Adaptmin* will have a lower C_{MAX}, and the *Adaptmin* will either always outperform or never do worse than *A* for families of instances described above.

IV. Simulation Results

We performed simulations to observe the performance of the discussed algorithms. Two criteria were observed, the efficiency of the schedule produced in terms of C_{AVE} and the computational complexity in terms of CPU time used. All simulations were performed using MATLAB®.

A. Adapt and Optimal Scheduling

We also compare *Adaptmin* and *A* to two more algorithms. *Adapt* is a variant of *Adaptmin*. When scheduling a job i using *Adapt*, the job is scheduled in the *first* interval among the first α_i intervals that has enough space to accommodate i instead of the interval containing the maximum empty space. However, if none of the first α_i intervals has space to accommodate

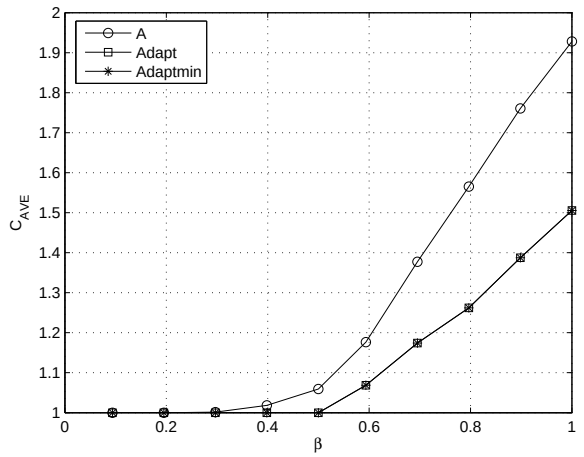

Fig. 4. Performance of *A*, *Adapt* and *Adaptmin* with $w = 2$

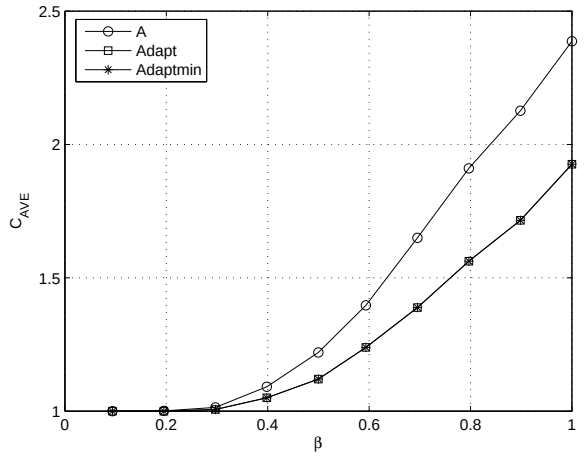

Fig. 5. Performance of *A*, *Adapt* and *Adaptmin* with $w = 3$

i, then like *Adaptmin*, *Adapt* expands intervals by the least amount required. More formally, the algorithm for *Adapt* can be obtained by replacing the step 5 in the algorithm for *Adaptmin* by "Starting with the first interval, try to schedule job i in the first $\alpha_i = \tau_i'/t'$ intervals.". One can think of *Adapt* as a 'first fit' solution. Simulation results show that performance of *Adapt* is very close to *Adaptmin*. However, since *Adapt* does not always search for the interval with maximum empty space, it is computationally more efficient than *Adaptmin*.

We also implemented a brute force algorithm which is optimal under the regime where all periods are a power of two and all periods are expanded by the same factor. This in some sense is the lower bound for performance of the other algorithms.

B. Efficiency of Algorithms

To compare the efficiency of algorithms, we plot C_{AVE} versus the requested bandwidth β. The input instance having a specific β can be generated in multiple ways. We outline our procedure, that tries to be as random as possible and not necessarily produce instances belong to families discussed in Section III-B. Since all algorithms round periods to nearest power of two, we restrict the periods to be power of two. The largest period is set to 128. We also restrict the job lengths to integers. For a given β, a point x_1 is randomly chosen between 0 and β. A support set of periods τ_i is generated such that $1/\tau_i \leq x_1$ and $\tau_i \leq 128$. From the support set, a period say τ_{1i} is randomly chosen and the maximum length b_{1i} is chosen such that $b_{1i}/\tau_{1i} \leq x$. Then a point say x_2 is randomly chosen from the interval $(b_{1i}/\tau_{1i}, \beta)$. A support set is again generated, such that $1/\tau_i \leq (x_2 - b_{1i}/\tau_{1i})$ and $\tau_i \leq 128$. A period say τ_{2i} is randomly chosen from the support set and the maximum length b_{2i} is chosen such that $b_{2i}/\tau_{1i} \leq (x_2 - b_{1i}/\tau_{1i})$. This process is repeated until the requested bandwidth for the job being generated is within $1/128$ of β. Note that the support set will always contain 128 as one of the periods, so the probability of jobs with period 128 is higher than other periods. We remedy this by truncating the support set to the smallest w periods and choosing randomly among them.

The simulation results are shown in Figure 3 through 6. Figure 3 shows the result when no truncation on the support set is performed, while figure 4 and 5 represent the case when $w = 2$ and $w = 3$, respectively. As expected, the C_{AVE} increases with β for all the algorithms. The curves of *Adapt* and *Adaptmin* exactly overlap in all the figures. However, overall *Adapt* performs slightly better than *Adaptmin*. Our algorithm clearly outperforms algorithm *A*. For example in figure 3 at $\beta = 0.5$, the average increase in periods for schedules produced using *A* is 0.61, while the same quantity for *Adapt* is 0.45, i.e., *Adapt* is around 25% more efficient. As w decreases, the difference between the algorithms increases further. Note that a smaller w on an average implies a smaller R, so with decreasing R, *Adapt* gives better performance. Figure 6 compares all the algorithms with the optimal scheduling with maximum period restricted to 32 instead of 128. As can be seen from the figure, our algorithms' perform quite close to the optimal scheduler.

C. Computational Complexity

The worst case computational complexity for all the algorithms being discussed is $O(nT'/t')$, however simulation results in this section show that they have quite different average case behavior.

To observe computational performance, instances were generated in a different manner compared to previous subsection. We studied computational complexity by varying two parameters, the number of jobs n and number of intervals, i.e., T'/t'. For a given n and T'/t', jobs were randomly generated and the computation time taken by *A*, *Adapt* and *Adaptmin* was measured. Computational time of the algorithms was measured

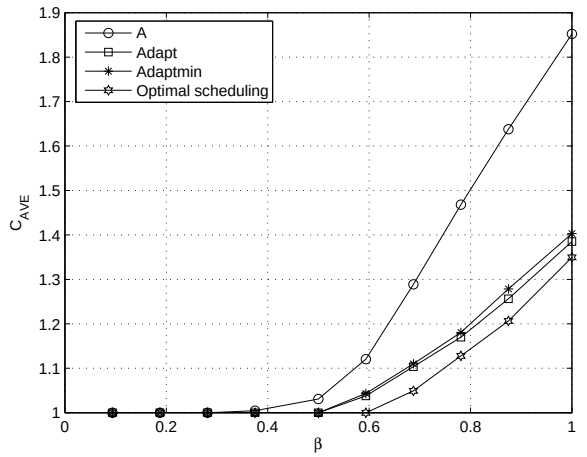

Fig. 6. Comparision of *A*, *Adapt*, *Adaptmin* with optimal scheduling

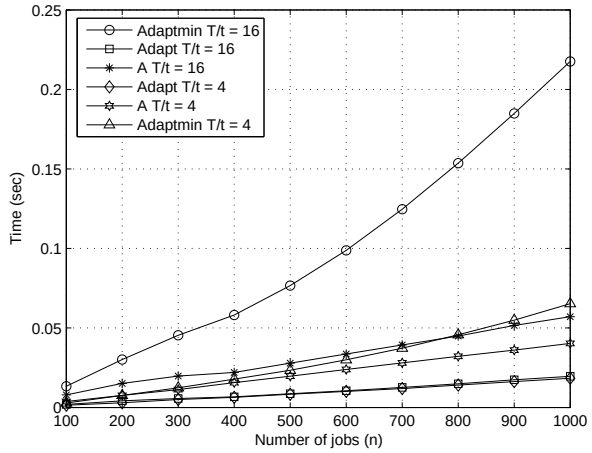

Fig. 7. Complexity comparison of *A*, *Adapt* and *Adaptmin*

using the "cputime" command of MATLAB®. The periods were again constrained to be power of two, while the lengths were required to be integral. The lengths for each job was chosen randomly between 1 and $\lfloor t/2 \rfloor$. The results have been plotted in Figure 7. When $T/t = 16$, T was equal to 128 and t was set to 8, while when $T/t = 4$, $T = 32$ and $t = 8$. Figure 7 shows that *Adapt* has a much better computational time compared to *A* and *Adaptmin*, while *A* does better than *Adaptmin*. But the difference among all the algorithms reduces with lower T/t ratio. Our tests show that the majority of algorithm *A*'s time is spent in delegating the jobs between the left and right child nodes in the tree that is constructed, while the majority of *Adaptmin* is spent in finding the interval with maximum empty space.

Fig. 7 also illustrates that both the algorithms are roughly linear with the number of jobs. However, *Adapt* seems quite insensitive to variation in T/t, while *A* and *Adaptmin* seem to be linear to the variation.

Overall, our results clearly indicate that both *Adapt* and *Adaptmin* produce much more efficient schedules than *A*. And from a computational complexity perspective *Adapt* is more efficient that *A*.

V. Conclusion

In this paper we proposed two algorithms *Adaptmin* and *Adapt* to perform perfectly periodic scheduling. We also developed upper bounds for *Adaptmin*. Simulation results show that both *Adaptmin* and *Adapt* perform much better than *A*, with *Adapt* performing slightly better than *Adaptmin*. We also compared *Adaptmin* and *Adapt* to optimal scheduling, the results show that our algorithms produce schedules close to optimal schedules. In terms of computational complexity, *Adapt* outperformed *A*.

Acknowledgment

The authors would like to thank Alejandro Icaza for the very fruitful discussions we had with him. We would also like to thank him for helping out with the simulations. We are also grateful to the anonymous reviewers for their insightful comments.

References

[1] M. H. Ammar and J. W. Wong, "The Design of Teletext Broadcast Cycles," *Performance Evaluation*, p. 235-242, December 1985.

[2] M. H. Ammar and J. W. Wong, "On the Optimality of Cyclic Transmission in Teletext Systems," *IEEE Transaction on Communication*, p. 1159-1170, November 1987.

[3] S. Anily, C. A. Glass and R. Hassin, "The Scheduling of Maintenance Service," *Discrete Applied Mathematics*, vol. 80, p. 27-42, 1998.

[4] S. Anily, C. A. Glass and R. Hassin, "Scheduling of Maintenance Services to Three Machines," *Annals of Operations Research*, vol. 86, p. 375-391, 1999.

[5] A. Bar-Noy, R. Bhatia, J. Naor and B. Schieber, "Minimizing Service and Operation Costs of Periodic Scheduling," *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, p.11-20, January 25-27, 1998.

[6] A. Bar-Noy, A. Nisgav and B. Patt-Shamir, "Nearly Optimal Perfectly-Periodic Schedules," *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, p.107-116, August 2001.

[7] A. Bar-Noy, V. Dreizin and B. Patt-Shamir, "Efficient Periodic Scheduling by Trees," *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, p. 23-27, June 2002.

[8] Z. Brakerski, A. Nisgav and B. Patt-Shamir, "General Perfectly Periodic Scheduling," *Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing*, p. 163-172, 2002.

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

[10] N. Dhanakoti, S. Gopalan and V. Sridhar, "Perfectly Periodic Scheduling for Fault Avoidance in IEEE 802.11e in the Context of Home Networks," *Proceedings of the 14th Annual IEEE International Symposium on Software Reliability Engineering*, November 17-20, 2003.

[11] R. Tijdeman, "The Chairman Assignment Problem," *Discrete Mathematics*, vol. 32, p. 323-330, 1980.

[12] W. Wei and C. Liu, "On a Periodic Maintenance Problem," *Operations Research Letters*, vol. 2, p.90-93, 1983.