Science of Computer Programming 172 (2019) 102–116

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Multi-core cyclic executives for safety-critical systems

Calvin Deutschbein [a], Tom Fleming [b], Alan Burns [b], Sanjoy Baruah [c,*]

[a] University of North Carolina at Chapel Hill, USA
[b] The University of York, UK
[c] Washington University in St. Louis, USA

A R T I C L E   I N F O

Article history:
Received 28 February 2018
Received in revised form 23 July 2018
Accepted 20 November 2018
Available online 30 November 2018

Keywords:
Multiprocessor scheduling
Cyclic executives
Linear program (LP)
Approximation algorithm

A B S T R A C T

In a cyclic executive, a series of pre-determined frames are executed in sequence; once the series is complete the sequence is repeated. Within each frame individual units of computation are executed, again in a pre-specified sequence. Although they suffer from a number of limitations, cyclic executives have the advantage of being fully deterministic, and may be implemented with very low runtime overhead; as a consequence of these advantages, run-time schedulers in highly safety-critical real-time systems have historically been implemented as cyclic executives.
Industrial applications of the cyclic executive framework are currently primarily restricted to uniprocessor platforms; in this paper, we consider the implementation of cyclic executives upon multi-core platforms. We present a Linear Programming (LP) based formulation of the problem of constructing cyclic executives upon multiprocessors for a particular kind of recurrent real-time workload — collections of implicit-deadline periodic tasks. We describe techniques for solving the LP formulation under different kinds of restrictions in order to obtain preemptive and non-preemptive cyclic executives. Our algorithms for constructing preemptive cyclic executives have running time polynomial in the size of the cyclic executive. We present an exact algorithm for constructing non-preemptive cyclic executives that has worst-case running time exponential in the size of the cyclic executive; however, state-of-the-art LP solvers appear to often be able to construct fairly large cyclic executives in a reasonable amount of time. We also present an approximation algorithm for constructing non-preemptive cyclic executives that does run in polynomial time, and evaluate the effectiveness of this approximation algorithm both theoretically via the speedup factor metric, and experimentally via experiments on synthetically generated workloads. We additionally identify a particular restricted kind of workload that is quite commonly found in practice, for which non-preemptive cyclic executives can be constructed more efficiently than in the general case.

1. Introduction and motivation

Real-time scheduling theory has made great advances over the past several decades. This includes the development of expressive models for representing real-time workloads (see, e.g., [1,2] for excellent surveys), comprehensive algorith-

* Corresponding author.
E-mail addresses: cd@cs.unc.edu (C. Deutschbein), tdf506@york.ac.uk (T. Fleming), alan.burns@york.ac.uk (A. Burns), baruah@wustl.edu (S. Baruah).

https://doi.org/10.1016/j.scico.2018.11.004
0167-6423/© 2018 Elsevier B.V. All rights reserved.

Table 1
Summary of cyclic-executive (CE) synthesis methodologies discussed in this paper. Both preemptive and non-preemptive CEs are considered. Preemptive CEs may be constructed exactly by formulating and solving an LP, or by constructing a graph and solving a network-flow problem on this graph. Non-preemptive CEs may be constructed exactly by formulating and solving an ILP, or approximately by solving a linear relaxation of this ILP.

Context	Representation	Method for schedule construction	Precision
Preemptive	Linear Program (LP)	From LP solution	Exact
Preemptive	Graph	Network-flow based	Exact
Non-preemptive	Integer LP (ILP)	From ILP solution	Exact
Non-preemptive	ILP	From solution to an LP-relaxation of ILP	Approximate

mic frameworks for scheduling, resource-allocation and synchronization in uniprocessor and multiprocessor environments, sophisticated techniques and insights enabling ever-more-efficient implementation of real-time systems, etc.

Despite all these advances, however, our interactions with industrial collaborators in highly safety-critical application domains, particularly those (such as avionics) that are subject to stringent certification requirements, reveal that the use of the *cyclic executive* approach [3,4] remains surprisingly wide-spread for scheduling safety-critical systems.

A cyclic executive is a simple deterministic scheme that consists, for a single processor, of the repeated execution of a series of *frames* (or *minor cycles* as they are often called). Each frame comprises a sequence of *jobs*. They execute in their defining sequence and must complete by the end of the frame. The set of frames is called the *major cycle*. Despite there being a number of drawbacks to using cyclic executives (some of which are discussed in Section 2), the cyclic executive approach offers two significant advantages, *predictability* and *low run-time overhead*, that are responsible for their widespread and continuing use for scheduling highly safety-critical systems (these advantages are also briefly discussed in Section 2).

Highly safety-critical real-time systems have traditionally been implemented upon custom-built processors that are designed to guarantee predictable timing behavior during run-time. Such processors are typically single-core to ensure predictability; hence most current techniques for constructing cyclic executives yield uniprocessor CEs. As safety-critical software has become ever more computation-intensive, however, it has proved too expensive to custom-built hardware powerful enough to accommodate the computational requirements of such software; hence, there is an increasing trend towards implementing safety-critical systems upon commercial off-the-shelf (COTS) platforms. Most COTS processors today tend to be multi-core ones; this motivates our research described here into the construction of CEs that are suitable for execution upon multi-core processors.

This research. We derive various approaches to constructing cyclic executives for implicit-deadline periodic task systems upon identical multiprocessor platforms; these approaches are summarized in Table 1. Many of these approaches share the commonality that they are based upon formulating the schedule construction problem as a linear program (LP). Cyclic executives in which jobs may be preempted can be derived from solutions to such LPs; since efficient polynomial-time algorithms are known for solving LPs, this approach enables us to design algorithms for constructing preemptive CEs that have running time polynomial in the size of the CE.

In order to construct non-preemptive CEs from a solution to the LP, the LP must be further constrained to require that some variables may only take on integer values: this is an *integer* linear program, or ILP. Solving an ILP is known to be NP-hard [5], and hence unlikely to be solvable exactly in polynomial time. Despite this inherent intractability of solving ILPs, however, the optimization community has recently been devoting immense effort to devise extremely efficient implementations of ILP solvers, and highly optimized libraries with such efficient implementations are widely available today. Modern ILP solvers, particularly when running upon powerful computing clusters, are often capable of solving ILPs with tens of thousands of variables and constraints. Since CEs are constructed prior to run-time, we believe that it is reasonable to attempt to solve ILPs exactly rather than only approximately, and seek to obtain ILP formulations *that we will seek to solve exactly* to construct non-preemptive multiprocessor CEs for implicit-deadline periodic task systems. However if this is not practical for particular problem instances, we devise an approximation algorithm with polynomial running time for constructing non-preemptive CEs, and evaluate the performance of this approximation algorithm vis-a-vis the exact one both via the theoretical metric of speedup factor, and via simulation experiments on synthetically generated workloads. We additionally show that for a particular kind of workload that is quite common in practice — systems of *harmonic* tasks – even better results are obtainable.

Organization. The remainder of this paper is organized as follows. In Section 2, we briefly describe the principles behind cyclic executives. In Section 3 we describe the periodic implicit-deadline task model, and introduce the terminology and notation that we use in this paper. In Section 4 we formulate the problem of constructing cyclic executives for implicit-deadline periodic tasks as a linear program; in Sections 5 and 6 respectively, we describe how such linear programming formulations may be solved to obtain preemptive and non-preemptive cyclic executives.

2. Cyclic executives

In this section we provide a brief introduction to the cyclic executive approach to hard-real-time scheduling. This is by no means comprehensive or complete; for a textbook description, please consult [6, Ch. 5.2–5.4].

Baker & Shaw [3] define a cyclic executive to be “a control structure [...] for explicitly interleaving the execution of several periodic processes. [...] The interleaving is done in a deterministic fashion so that execution timing is predictable.” In the cyclic executive approach, a schedule called a *major schedule* is determined prior to run-time, which describes the sequence of actions (i.e., computations) to be performed during some fixed period of time called the *major cycle*. The actions of a major schedule are executed cyclically, going back to the beginning at the start of each major cycle.[1] The major schedule is further divided into one or more *frames* (also known as minor schedules or minor cycles). As stated in [7], a *common way to describe a schedule is to describe a complete major cycle as a sequence of different minor cycles, and to express each minor cycle as a sequence* of actions. Each frame is allocated a fixed length of time during which the computations assigned to that frame must be executed. Timing correctness is monitored at frame boundaries via hardware interrupts generated by a timer circuit: if the computations assigned to a frame are discovered to have not completed by the end of the frame then a *frame overrun* error is flagged and control transferred to an error-handling routine.

The chief benefits of the cyclic executive approach to scheduling are its implementation simplicity and efficiency, and the timing predictability it offers: if we have a reliable upper bound on the execution duration of each computation then an application's schedulability is determined by construction (i.e., if we are successful in building the CE then we can be assured that all deadlines are met).

The chief challenge lies in constructing the schedules; i.e., in defining the frame durations and determining the actions to be performed within each frame. The schedule construction problem is rendered particularly challenging by the requirement that for implementation efficiency considerations, timing monitoring is performed only at frame boundaries — as stated above, a timer is set at the start of a frame to go off at the end of the frame, at which point in time it is verified that all actions assigned to that frame have indeed completed execution (if not, corrective action must be taken via a call to error-handling routines). CEs are typically used for periodic workloads only (rather than for mixes of periodic and sporadic tasks). Hence the schedule-generation approach proposed in [3] requires that at least one frame lie within the interval formed by the instants that each action — "job" — become available for execution, and the instant that it has a deadline. For efficiency considerations, it is usually required that all tasks have a period that is a multiple of the minor cycle, and a deadline that is no smaller than the minor cycle duration. Schedule construction is in general highly intractable for many interesting models of periodic processes [3]; however, heuristics have been developed that permit system developers to construct such schedules for reasonably complex systems (as Baker & Shaw have observed [3], "if we do not insist on optimality, practical cases can be scheduled using heuristics").

In this paper, we model our periodic workload as a task system of implicit-deadline periodic tasks. Some of our results additionally require that the tasks have harmonic periods: for any pair of tasks τ_i and τ_j, it is the case that T_i divides T_j exactly or T_j divides T_i exactly. Although this does constitute a significant restriction on the periodic task model, many safety-critical systems appear to respect this restriction.

Current practice. Current approaches to the construction of CEs in safety-critical systems are often *ad hoc* and based on the expertise of individual system integrators who have built up years of experience in synthesizing such CEs. This approach often requires that additional restrictions (such as harmonic periods) be placed upon the workload. The academic and research community appears to have not devoted too much effort in this direction. A few tools have been developed that are primarily based on exhaustive search incorporating heuristic rules for optimizing the direction of the search [8,9], specializing search heuristics such as simulated annealing [10], and model checking [11]; the performance of these tools generally do not scale well with problem size. To our knowledge, no tools for constructing CEs have been developed that exploit the recent significant advances in the state of the art of linear programming-based optimization tools.

3. Workload model

Throughout this paper we assume that we are given a task system $\tau = \{\tau_i = (C_i, T_i)\}_{i=1}^{N}$ of N implicit-deadline periodic[2] tasks that are to be scheduled upon an m-processor identical multiprocessor platform. The worst-case execution time (WCET) of τ_i is C_i, and its period is T_i. Let P denote the least common multiple (lcm) of the periods of all the tasks in τ (P is often called the *hyper-period* of τ), and let F denote the greatest common divisor (gcd) of the periods of all the tasks in τ. P is selected as the duration of the major cycle, and F the duration of the minor cycle, of the CEs we will construct (Fig. 1).

Some further notation and terminology: Let $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ denote all the jobs generated by τ that have their arrival times and deadlines within the interval $[0, P)$, and let a_i, c_i and d_i denote the arrival time, WCET, and (absolute) deadline respectively of job j_i. (We will often represent a job j_i by an ordered 3-tuple of its parameters: $j_i \overset{\text{def}}{=} (a_i, c_i, d_i)$. We refer to the interval $[a_i, d_i)$ as the *scheduling window* of this job j_i; in any correct schedule, each job j_i will receive at least c_i units of execution during its scheduling window.) Note that the number of jobs n may in general take on a value that is exponential in the number of tasks N. Since we are seeking to explicitly construct a schedule for the n jobs, we argue that it is reasonable to evaluate the efficiency of algorithms for constructing these schedules in terms of the number of jobs n to be scheduled rather than in terms of the number of periodic tasks N. (Indeed, the *size* of the CE that is pre-computed and

[1] Multiple major schedules may be defined for a single system, specifying the desired system behavior for different *modes* of system operation; switching between modes is accomplished by swapping the major schedule used. If a major cycle is of not too large a duration, then switches between modes may be restricted to only occur at the end of major cycles.

[2] We highlight that these are periodic, not sporadic, tasks: τ_i generates jobs at time-instants $k \times T_i$, for all $k \in \mathbb{N}$.

N and m	Number of **tasks** and **processors**
$\tau_i = (C_i, T_i)$	The i'th task has worst-case execution time C_i and period T_i
P	$\text{lcm}_{i=1}^{N}\{T_i\}$ — the *hyperperiod*. Selected as major cycle duration
F	$\text{gcd}_{i=1}^{N}\{T_i\}$. Selected as minor cycle (frame) duration
f	The amount of execution that a single processor can accommodate in one frame. Upon unit-speed processors, $f = F$
Φ_k	The k'th frame, for $k \in \{1, 2, \ldots, P/F\}$
n	The total number of *jobs* in one hyperperiod. $n = \sum_{i=1}^{N}(P/T_i)$
$j_i = (a_i, c_i, d_i)$	The i'th job, $1 \le i \le n$. Its arrival time, WCET, and absolute deadline
$\mathcal{J}$	The collection of these n jobs
x_{ijk}	LP variable: the fraction of the i'th job assigned to the j'th processor during the k'th frame

Fig. 1. Some of the notation used in this paper.

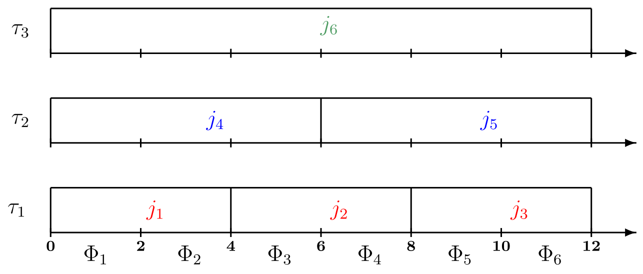

Fig. 2. The jobs generated by the task system of Example 1.

stored for use during run-time is polynomial in n rather than N; hence, we believe it makes sense to consider algorithms that have run-time polynomial in the size of the generated object to be "efficient".)

Without loss of generality, we assume that the tasks are indexed according to non-decreasing periods: $T_i \le T_{i+1}$ for all i, $1 \le i < N$. For *harmonic* task systems τ, the tasks have harmonic periods: T_i divides T_{i+1} exactly for all i, $1 \le i < N$.

Example 1. Consider a system τ comprising three tasks τ_1, τ_2, and τ_3, with periods $T_1 = 4$, $T_2 = 6$, and $T_3 = 12$. $P = \text{lcm}(4, 6, 12) = 12$; $F = \text{gcd}(4, 6, 12) = 2$. Therefore, minor cycle duration is 2, major cycle duration is 12. For this τ, $\mathcal{J}$ comprises the six jobs j_1–j_6 depicted in Fig. 2. As shown in the figure, task τ_1 generates jobs j_1, j_2, and j_3, task τ_2 generates jobs j_4 and j_5, and task τ_3 generates job j_6. There are $(12/2) =$ six frames or minor cycles within the major cycle — these are labeled in the figure as $\Phi_1, \Phi_2, \ldots, \Phi_6$ with Φ_k spanning the interval $[2(k-1), 2k]$. □

4. Representing cyclic executives as linear programs

In this section we represent the problem of constructing a cyclic executive as a linear program. We start out in Section 4.1 with a brief review of some well-known facts concerning linear programs that we will use in later sections of the paper; the LP representation of CEs is in Section 4.2.

4.1. Some linear programming background

In an integer linear program (ILP), one is given a set of v variables, some or all of which are restricted to take on integer values only, a collection of "constraints" that are expressed as *linear* inequalities over these v variables, and an "objective function," also expressed as a linear inequality of these variables. The set of all points in v-dimensional space over which all the constraints hold is called the *feasible region* for the integer linear program. The goal is to find the extremal (maximum or minimum, as specified) value of the objective function over the feasible region.

A linear program (LP) is like an ILP, without the constraint that some of the variables are restricted to take on integer values only. That is, in an LP over a given set of v variables, one is given a collection of constraints that are expressed as linear inequalities over these v variables, and an objective function, also expressed as a linear inequality of these variables. The region in v-dimensional space over which all the constraints hold is again called the feasible region for the linear program, and the goal is to find the extremal value of the objective function over the feasible region. A region is said to be *convex* if, for any two points $\mathbf{p_1}$ and $\mathbf{p_2}$ in the region and any scalar $\lambda, 0 \le \lambda \le 1$, the point $(\lambda \cdot \mathbf{p_1} + (1-\lambda) \cdot \mathbf{p_2})$ is also in the region. A *vertex* of a convex region is a point $\mathbf{p}$ in the region such that there are no distinct points $\mathbf{p_1}$ and $\mathbf{p_2}$ in the region, and a scalar $\lambda, 0 < \lambda < 1$, such that $[\mathbf{p} \equiv \lambda \cdot \mathbf{p_1} + (1-\lambda) \cdot \mathbf{p_2}]$.

It is known that an LP can be solved in polynomial time by the ellipsoid algorithm [12] or the interior point algorithm [13]. We do not need to understand the details of these algorithms: for our purposes, it suffices to know that LP problems can be efficiently solved (in polynomial time).

We now state without proof some basic facts concerning linear programming optimization problems.

Fact 1. The feasible region for a LP problem is convex, and the objective function reaches its optimal value at a vertex point of the feasible region.

An optimal solution to an LP problem that is a vertex point of the feasible region is called a **basic solution** to the LP problem.

Fact 2. A basic solution to an LP can be found in polynomial time.

Fact 3. Consider a linear program on v variables with each variable subject to the constraint that it be ≥ 0 (such constraints are called non-negativity constraints). Suppose that in addition to these non-negativity constraints there are c other linear constraints. If $c < v$, then *at most c of the variables have non-zero values* at each vertex of the feasible region (including at all basic solutions).

4.2. An LP representation of CEs

Given a periodic task system comprising N tasks for which an m-processor cyclic executive is to be obtained, we now describe the construction of a linear program with $\left(N \times m \times \frac{P}{F}\right)$ variables, each of which is subject to a non-negativity constraint (i.e., each may only take on a value ≥ 0), and $\left(n + (m+N) \times \frac{P}{F}\right)$ additional linear constraints (here n denotes the total number of jobs generated by the task system over one hyper-period, P denotes the least common multiple of the period of the tasks, and F denotes the greatest common divisor of the periods of the tasks).

§1. Variables. We will have a variable x_{ijk} denote the fraction of job j_i that is scheduled upon the j'th processor during the k'th frame. The index i takes on each integer value in the range $[1, n]$ (recall that n denotes the total number of jobs generated by all the periodic tasks over the hyper-period). For each i,

- The index j takes on each integer value in the range $[1, m]$ — here m denotes the number of processors.
- Note that job j_i may only execute within those frames that are contained in the scheduling window — the interval $[a_i, d_i)$ — of job j_i. The index k, therefore, only takes on values over the range of frame-numbers of those frames contained within $[a_i, d_i)$.
  With regard to the example task system of Example 1 as depicted in Fig. 2, for instance, job j_3 may only execute during the 5'th or the 6'th frames (Φ_5 and Φ_6). Hence, assuming a 2-processor platform ($m = 2$), the variables defined for job j_3 are $x_{315}, x_{316}, x_{325}$, and x_{326}.

The total number of x_{ijk} variables is equal to $[N \times m \times (P/F)]$, where N denotes the number of periodic tasks (*not* the number of jobs), m denotes the number of processors, and P/F represents the number of minor cycles.

Example 2. For the task system of Example 1 on two processors, the x_{ijk} variables corresponding to jobs of task τ_1 are

$$\overbrace{x_{111},\ x_{121},\ x_{112},\ x_{122}}^{j_1},\quad \overbrace{x_{213},\ x_{223},\ x_{214},\ x_{224}}^{j_2},\quad \overbrace{x_{315},\ x_{325},\ x_{316},\ x_{326}}^{j_3}$$

Those corresponding to jobs of task τ_2 are

$$\overbrace{x_{411},\ x_{421},\ x_{412},\ x_{422},\ x_{413},\ x_{423}}^{j_4},\quad \overbrace{x_{514},\ x_{524},\ x_{515},\ x_{525},\ x_{516},\ x_{526}}^{j_5}$$

Corresponding to the sole job of τ_3, we have the twelve variables

$$\overbrace{x_{611},\ x_{621},\ x_{612},\ x_{622},\ x_{613},\ x_{623},\ x_{614},\ x_{624},\ x_{615},\ x_{625},\ x_{616},\ x_{626}}^{j_6}$$

for a total of 36 variables. □

§2. An objective function. Since LP's allow us to specify an objective function to optimize (minimize/maximize), let us seek to minimize the speed or computing capacity that the processors must possess in order for there to exist a feasible cyclic executive. Specifically, let f denote the amount of computing that can be accomplished by a processor executing for the duration F of an entire frame; if the processors were to be of unit speed, then $f = F$. We will define the following objective function for our LP:

$$\textbf{minimize } f \tag{1}$$

This is equivalent to determining the minimum-speed processors needed in order to construct a feasible CE. That is, the minimum value of f that is obtained upon solving the LP represents the minimum amount of computation that is needed to be completed by an individual processor within a duration F; if the available processors can indeed accommodate this amount of computation within a single frame, then the solution is a feasible one.

§3. Constraints. Since the x_{ijk} variables represent fractions of jobs, they must all be assigned values that are ≥ 0; hence, they are all subject to non-negativity constraints. In addition, the $\left(N \times m \times \frac{P}{F}\right)$ variables defined above are used to construct a linear program representation of a CE, via the following constraints:

1. We represent the requirement that each job must receive the required amount of execution by having the constraints

$$\sum_{\text{all } j,k} x_{ijk} = 1 \text{ for each } i,\ 1 \leq i \leq n \tag{2}$$

   There are n such constraints, one per job.
2. We represent the requirement that each processor may be assigned no more than f units of execution during each minor cycle by having the constraints

$$\sum_{\text{all } i} x_{ijk} \cdot c_i \leq f \text{ for each } j,\ 1 \leq j \leq m \text{ and } k,\ 1 \leq k \leq P/F \tag{3}$$

   There are $m \times (P/F)$ such constraints.
3. We represent the requirement that each job may be assigned no more than f units of execution during each minor cycle by having the constraints

$$\sum_{\text{all } j} x_{ijk} \cdot c_i \leq f \text{ for each } i,\ 1 \leq i \leq n \text{ and } k,\ 1 \leq k \leq P/F \tag{4}$$

   There are $N \times (P/F)$ such constraints (note that N denotes the number of periodic tasks, *not* the number of jobs).

The total number of constraints is thus equal to $[n + (m + N) \times (P/F)]$.

Example 3. For the task system of Example 1 on two processors, there will be a total of 36 constraints. Rather than enumerate all 36, we illustrate one constraint of each kind (Constraints 2, 3, and 4).

- Constraint 2, instantiated for job j_1, would be

$$x_{111} + x_{121} + x_{112} + x_{122} = 1$$

- Constraint 3, instantiated for the second processor and the fourth frame Φ_4, would be of the form $\sum_i x_{i24}$, with the index i ranging over all those jobs for which the scheduling windows overlap with frame Φ_4. These are jobs j_2 of task τ_1, job j_5 of task τ_2, and job j_6 of task τ_3. The constraint is therefore

$$x_{224} \times C_1 + x_{524} \times C_2 + x_{624} \times C_3 \leq f$$

  Here, we are using the fact that since job j_2 (jobs j_5 and j_6, respectively) is generated by task τ_1 (tasks τ_2 and τ_3, resp.), $c_2 \leftarrow C_1$ ($c_4 \leftarrow C_2$ and $c_6 \leftarrow C_3$, resp.).
- Constraint 4, instantiated for job j_4 and the third frame Φ_3, is

$$x_{413} \times C_2 + x_{423} \times C_2 \leq f$$

  Here, we are using the fact that since job j_4 is generated by task τ_2, $c_5 \leftarrow C_2$. □

§3. Solving the LP. We saw above how a cyclic executive for N implicit-deadline periodic tasks upon m processors could be represented as the solution to a linear program with

$$\left(N \times m \times \frac{P}{F}\right) \text{ variables and } \left(n + (m + N) \times \frac{P}{F}\right) \text{ constraints,}$$

where n denotes the total number of jobs generated by the task system over one hyper-period, P denotes the least common multiple of the period of the tasks, and F denotes the greatest common divisor of the periods of the tasks. Observe that

1. Given an assignment of integer values (i.e., either 0 or 1) to each of the x_{ijk} variables that satisfy the constraints of the LP, we may construct a non-preemptive cyclic executive in the following manner: for each x_{ijk} that is assigned the value 1, schedule the execution of job j_i on the j'th processor during the k'th frame.
2. Given an assignment of non-negative values to the x_{ijk} variables that satisfy the constraints of the LP, we may construct a global preemptive cyclic executive in the following manner. For each x_{ijk} that is assigned a non-zero value, schedule job j_i for a duration $x_{ijk} \times c_i$ on the j'th processor during the k'th frame. (Of course, care must be taken to ensure that during each frame no job executes concurrently upon two different processors — we will see in Section 5 below how this is ensured.)

That is, an integer solution to the ILP yields a non-preemptive cyclic executive while a fractional solution yields a global preemptive cyclic executive. We discuss the problem of obtaining such solutions, and thereby obtaining preemptive and non-preemptive cyclic executives respectively, in Sections 5 and 6 respectively.

5. Preemptive cyclic executives

In this section we discuss the problem of constructing preemptive cyclic executives for implicit-deadline periodic task systems by obtaining solutions to the linear program described above.

Let us suppose that we have solved the linear program, and have thus obtained an assignment of non-negative values to the x_{ijk} variables that satisfy the constraints of the LP. We now describe the manner in which we construct a preemptive cyclic executive for the k_o'th frame Φ_{k_o}; the entire cyclic executive is obtained by repeating this procedure for each k_o, $1 \le k_o \le (P/F)$.

For each job j_{i_o} observe that

$$\chi_{i_o} \overset{\text{def}}{=} \sum_{j=1}^{m} x_{i_o j k_o}$$

represents the total amount of execution assigned to job j_{i_o} during frame Φ_{k_o} in the solution to the LP. By Constraint 4 of the LP, it follows that $\chi_{i_o} \le f$ for each job j_{i_o}; i.e. no job is assigned more than f units of execution over the frame. Additionally, it follows from summing Constraint 3 of the LP over all m processors (i.e., for all values of the variable j in Constraint 3) that

$$\left(\sum_{i_o=1}^{n} \chi_{i_o}\right) \le m \times f.$$

We have thus shown that **(i)** no individual job is scheduled during the frame for more than the computing capacity of a single processor during one frame, and **(ii)** the total amount of execution scheduled over the interval does not exceed the cumulative computing capacity of the frame (across all m processors). We may therefore construct a schedule within the frame using McNaughton's wrap-around rule [14] in the following manner:

1. We order the jobs that receive any execution within frame Φ_{k_o} arbitrarily.
2. Then we begin placing jobs on the processors in order, filling the j'th processor entirely before starting the $(j+1)$'th processor. Thus, a job j_{i_o} may be split across processors, assigned to the last t time units of the frame on the j'th processor and the first $(\chi_{i_o} - t)$ time units of the frame on the $(j+1)$'th processor; since $\chi_{i_o} \le f$, these assignments will not overlap in time.

It is evident that this can all be accomplished efficiently within run-time polynomial in the representation of the task system.

Implementation. In Section 6.2.1 below, we describe experiments that we have conducted comparing ILP-based exact and LP-based approximate algorithms for constructing *non*-preemptive CEs. These experiments required us to solve LPs, similar to the kind described here, using the Gurobi Optimization tool [15]; performance of the Gurobi Optimization tool scaled very well with the size of the task system in these experiments. (We also point out that even when not asked for integer solutions, the Gurobi Optimization tool typically tends to find non-preemptive solutions if they exist; hence very often a non-preemptive CE is obtained even when not asked for!)

5.1. A network-flow approach

As we saw in Section 4 above, the Linear Programming framework offers an attractive abstraction for formalizing the requirements of cyclic executives for implicit-deadline periodic task systems. Given such an LP formulation of a cyclic executive, we are able to obtain both preemptive or non-preemptive schedules by solving the linear program under different

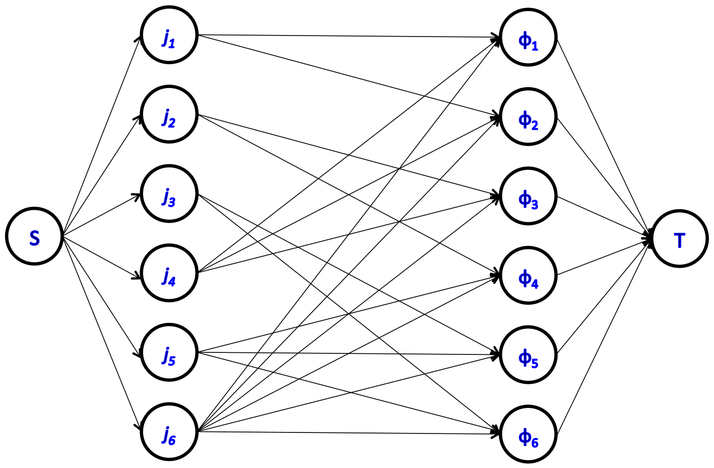

Fig. 3. The network flow graph for the task system of Example 1 (as depicted in Fig. 2). The capacity of each edge $(\mathbf{S}, j_i) \in E_1$ is equal to j_i's WCET c_i. The capacity of each edge in E_2 is equal to f; the capacity of each edge in E_3 is $(m \times f)$.

constraints (i.e., integer-valued solutions only, versus real-valued solutions); above, we saw how we can obtain a preemptive cyclic executive from a real-valued solution.

For obtaining preemptive cyclic executives, there is an alternative (i.e., not LP-based) algorithm that we briefly describe in this section. This algorithm is based upon first constructing a network (a directed graph with capacity constraints specified upon its edges) from the specifications of a given task system, and establishing equivalence between a preemptive cyclic executive for this task system and a flow of a certain size between particular nodes in the graph. Such an equivalence immediately yields the polynomial-time cyclic-executive construction algorithm we desire, since algorithms are known for determining network flows in polynomial time.[3]

As before, let us suppose that we are given a task system comprising N implicit-deadline periodic tasks for which we seek to construct a cyclic executive upon m unit-speed processors. We now describe below the construction of a weighted digraph G. (The graph obtained by applying this construction to the example task system of Fig. 2 is depicted in Fig. 3; the reader may wish to refer to Fig. 3 to help understand the construction described here.) Graph G is a "layered" graph: the vertex set of G is the union of 4 disjoint sets of vertices $V_1, \ldots, V_4$, and the edge set of G is the union of 3 disjoint sets of edges $E_1, \ldots, E_3$, where E_i is a subset of $(V_i \times V_{i+1})$, $1 \le i \le 3$. G is thus a 4-layered graph, in which all edges connect vertices in adjacent layers. G is constructed so that the sets of vertices are as follows:

1. $V_1 = \{\mathbf{S}\}$
2. V_2 has n vertices, one corresponding to each job generated over the hyper-period
3. V_3 has P/F vertices, one corresponding to each frame defined over the hyper-period, and
4. $V_4 = \{\mathbf{T}\}$

The sets of edges of G are as follows:

1. E_1 comprises n edges: there is an edge from **S** to each of the n vertices in V_2. The capacity of the edge connecting **S** to each vertex $u \in V_2$ is equal to WCET of the job corresponding to the vertex u.
2. E_2 comprises $(N \times (P/F))$ edges: there is a single edge of capacity f from each vertex $u \in V_2$ to each vertex $v \in V_3$ that corresponds to a frame contained within the scheduling window of the job corresponding to the vertex u.
3. E_3 comprises (P/F) edges: there is an edge of capacity $m \times f$ from each vertex in V_3 to vertex **T**.

Recall that c_i denotes the WCET of job j_i; therefore, $\left(\sum_{i=1}^{n} c_i\right)$ equals the sum of the WCETs of all the jobs of all the tasks over one hyper-period. The following lemma establishes the equivalence between a preemptive cyclic executive for the task system and a flow of this size between the vertices **S** and **T** in the graph constructed above.

Lemma 1. *Each flow of size $\left(\sum_{i=1}^{n} c_i\right)$ from vertex* **S** *to vertex* **T** *in the directed graph G corresponds to a preemptive m-processor cyclic executive for the task system.*

Proof. We first show that any cyclic executive can be represented as a flow of size n.

[3] This approach of reducing a preemptive scheduling problem to a network-flow one is fairly standard in the scheduling literature; see, e.g., [16,17] for prior example applications of its use.

- Assign a flow from vertex **S** to each vertex $u \in V_1$ of size equal to the WCET of the job corresponding to vertex u. Thus, there is a flow of the desired size leaving vertex **S**.
- Suppose that the job j_i is scheduled for a duration x in frame Φ_k in the cyclic executive. Assign a flow of size x from the vertex in V_2 corresponding to j_i, to the vertex in V_3 corresponding to Φ_k.
  Observe that since each job j_i is scheduled for its WCET in the cyclic executive, this construction ensures that the flow into each vertex in V_2 flows out of that vertex.
- Suppose that the total WCET of jobs scheduled during the frame Φ_k in the cyclic executive equals W; assign a flow of size W from the vertex in V_3 corresponding to frame Φ_k to the vertex **T**.
  Observe that the total amount of execution scheduled upon the m processors during any frame is at most $m \times f$; hence the capacity constraint on the edges in E_3 are satisfied.

Next, we show that given a flow of size $\left(\sum_{i=1}^{n} c_i\right)$ in the graph, we can construct an m-processor preemptive cyclic executive for the task system. The schedule is constructed using the following scheduling rule:

For each edge $(u, v) \in V_2 \times V_3$ that has a flow of size x, schedule the job corresponding to $u \in V_2$ for a duration x in the frame corresponding to $v \in V_3$.

We now argue that this rule does indeed yield a correct cyclic executive:

- Since the flow is of the desired size, there must be a flow from vertex **S** to each vertex in $u \in V_2$ of size equal to the WCET of the job corresponding to u. Further, this flow into each vertex of V_2 must exit the vertex via edges in E_2. Therefore, each of the n jobs does get scheduled for a duration equal to its WCET upon frames within its scheduling window by the above scheduling rule.
- It remains to show that the schedule constructed by the scheduling rule is a feasible one: we do so by the following argument. Since each edge in E_2 is of capacity f, no individual job is scheduled for a duration $> f$ within any single frame. Since each edge in E_3 has a capacity of $m \times f$, the total amount of assigned execution within any single frame does not exceed the available capacity in that frame across all m processors. McNaughton's wrap-around rule [14] schedule may therefore be applied for each frame to construct the schedule within the frame. $\square$

Lemma 1 above shows that in order to construct a preemptive cyclic executive, it suffices to determine whether a flow of a specified size exists in the graph G. The Ford–Fulkerson network-flow algorithm [18] can determine the largest flow in G polynomial time. We can therefore apply the Ford–Fulkerson network-flow algorithm to the constructed graph to determine the largest flow from vertex **S** to vertex **T** in the graph G; if this flow is of size equal to the sum of the WCETs of all the jobs, then we apply the scheduling rule described in the proof of Lemma 1 to construct the cyclic executive.

6. Non-preemptive cyclic executives

Synthesizing non-preemptive cyclic executives is highly intractable: it follows directly from results in [19] that synthesizing such CEs is NP-hard in the strong sense even for highly-restricted instances in which all tasks have the same period. In this section, we discuss the process of obtaining 0/1 integer solutions to the linear program defined in Section 4.2; as discussed there, such a solution can be used to construct non-preemptive cyclic executives for the periodic task system represented using the linear program.

Let us start out observing that in order for a non-preemptive cyclic executive to exist, it is necessary that any job fits into an individual frame; i.e., that

$$\max_{i=1}^{N}\{C_i\} \le f \tag{5}$$

Any task system for which this condition does not hold cannot be scheduled non-preemptively.[4]

Let us now take a closer look at the LP that was constructed in Section 4.2. Consider any 0/1 integer solution to this LP. Each x_{ijk} variable will take on value either zero or one in such a 0/1 integer solution; hence in the LP, the *Constraints 2 render the Constraints 4 redundant*. To see why this should be so, consider any job (say, j_{i_o}), and any frame (say, Φ_{k_o}). From Constraints 2 and the fact that each x_{ijk} variable is assigned a value of zero or one, it follows that in any 0/1 integer solution to the linear program we will have

$$\left(\sum_j x_{i_o j k_o} = 0\right) \quad \text{or} \quad \left(\sum_j x_{i_o j k_o} = 1\right),$$

[4] The problem of scheduling such systems by permitting a controlled amount of *splitting* of individual jobs is explored, in a somewhat wider context, in [20].

depending upon whether job j_{i_o} is scheduled (on any processor) within the frame Φ_{k_o} or not. We thus see that at most one of the $x_{i_o jk_o}$'s can equal 1, from which it follows that Constraint 4 necessarily holds for job j_{i_o} within the frame Φ_{k_o}. We may therefore omit the Constraints 4 in the linear program. Hence for non-preemptive schedules, we have a somewhat simpler ILP that needs to be solved, comprising

$$\left(N \times m \times \frac{P}{F}\right) \text{ variables but only } \left(n + m \times \frac{P}{F}\right) \text{ constraints.}$$

For our example task system, this would translate to 36 variables (as before), but just 12 constraints (rather than the 36 constraints discussed in Example 3).

6.1. An approximation algorithm

The problem of finding a 0/1 solution to a Linear Program is NP-hard in the strong sense; all algorithms known for obtaining such solutions have running time that is exponential in the number of variables and constraints. As we had mentioned earlier, this intractability of Integer Linear Programming does not necessarily rule out the ILP-based approach to constructing cyclic executives that we have described above, since excellent solvers have been implemented that are able to solve very large ILPs in reasonable amounts of time.

However, the fact of the matter is that not all ILPs can be solved efficiently. We now describe an approximation algorithm for constructing Cyclic Executives, that does not require us to solve ILPs exactly. The algorithm is approximate in the sense that it may fail to construct Cyclic Executives for some input instances for which CEs do exist (and could have been constructed using the exponential-time ILP-based method discussed above). In Theorem 1 we quantify the non-optimality of our approximation algorithm.

Our algorithm starts out constructing the linear program as described in Section 4.2, but without the Constraints 4 (as discussed above, the Constraints 2 render these redundant). However, rather than seeking to solve the NP-hard problem of obtaining a 0/1 integer solution to this problem, we instead replace the 0/1 integer constraints with the requirement that each x_{ijk} variable be a non-negative real number no larger than one (i.e., that $0 \le x_{ijk} \le 1$ for all variables x_{ijk}), and then obtain a basic solution[5] to the resulting linear program (without the constraint that variables take on integer values). As stated in Fact 2 of Section 4.1, such a basic solution can be found efficiently in polynomial time.

Recall that our LP has $\left(N \times m \times \frac{P}{F}\right)$ variables but only $\left(n + m \times \frac{P}{F}\right)$ constraints. By Fact 3 of Section 4.1, at most $\left(n + m \times \frac{P}{F}\right)$ of the variables will take on non-zero values at the basic solution. Some of these non-zero values will be equal to one — each such value determines the frame and processor upon which a job is to be scheduled in the cyclic executive. I.e., *for each x_{ijk} that is assigned a value equal to one in the basic solution, we assign job j_i to the j'th processor during frame Φ_k.*

It remains to schedule the jobs which were not assigned as above — these are the jobs for which Constraint 2 was satisfied in the LP solution by having multiple non-zero terms on the LHS. This is done according to the following procedure; the correctness of this procedure is proved in [21].

1. Consider all the variables $X \stackrel{\text{def}}{=} x_{ijk}$ that have been assigned non-zero values strictly less than one in the basic solution. That is,
$$X \stackrel{\text{def}}{=} \left\{x_{ijk} \text{ such that } 0 < x_{ijk} < 1 \text{ in the basic solution}\right\}$$
2. Construct a bipartite graph with
   (a) A vertex for each job j_{i_o} such that there is some (one or more) $x_{i_o jk} \in X$. Let V_1 denote the set of all such vertices that are added.
   (b) A vertex for each ordered pair $[j_o, k_o]$ such that there is some (one or more) $x_{ij_o k_o} \in X$. Let V_2 denote the set of all such vertices that are added.
   (c) For each $x_{i_o j_o k_o} \in X$ add an edge in this bipartite graph from the vertex in V_1 corresponding to job j_{i_o}, to the vertex in V_2 corresponding to ordered pair $[j_o, k_o]$.
3. It has been shown in [21] that there is a matching in this bipartite graph that includes all the vertices in V_1. Such bipartite matchings can be found in polynomial time using network-flow algorithms [22,18] (see [23, Chapter 26.3] for a textbook description).
4. Once such a bipartite matching is obtained, each job corresponding to a vertex in V_1 is assigned to the processor and frame corresponding to the vertex in V_2 to which it has been matched. In this manner, each processor in each frame is guaranteed to be assigned at most one job during this process of assigning the jobs that were not already assigned in the basic solution.

[5] Recall from Section 4.1 above that a *basic solution* to an LP is an optimal solution that is a vertex point of the feasible region defined by the constraints of the LP.

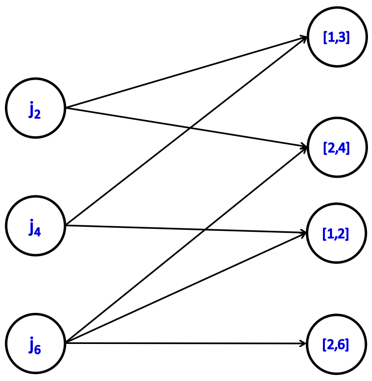

Fig. 4. The bipartite graph corresponding to a possible LP solution for the task system of Example 1 (as depicted in Fig. 2).

Example 4. Let us re-visit the example task system depicted in Fig. 2. Suppose that we have constructed and solved a Linear Program corresponding to this task system, and that this solution assigned non-integer values to only the following variables:

$$\overbrace{x_{213} \leftarrow 0.5;\ x_{224} \leftarrow 0.5;}^{j_2}\quad \overbrace{x_{424} \leftarrow 0.7;\ x_{412} \leftarrow 0.3;}^{j_4}\quad \overbrace{x_{624} \leftarrow 0.8;\ x_{612} \leftarrow 0.1;\ x_{626} \leftarrow 0.1}^{j_6}$$

1. We would construct vertices corresponding to jobs j_2, j_4, and j_6 in V_1, and corresponding to the ordered pairs $[1,3], [2,4], [1,2]$, and $[2,6]$ in V_2; the resulting bipartite graph is depicted in Fig. 4.
2. A possible matching in this graph pairs j_6 with $[2,6]$, j_4 with $[1,2]$, and j_2 with $[1,3]$.
3. Based on this matching, we would assign j_6 entirely to the second processor during frame Φ_6, j_4 entirely to the first processor during frame Φ_2, and j_2 entirely to the first processor during frame Φ_3.

As can be seen, in each frame any processor gets at most one additional job assigned to it. □

6.2. Evaluating the approximation algorithm

We now compare the effectiveness of the polynomial-time approximation algorithm of Section 6.1 with that of the ILP-based exact algorithm (solving which takes exponential time in the worst case). We start out with theoretical evaluation: Corollary 1 quantifies the worst-case performance of the approximation algorithm via the speedup factor metric. We have also conducted some simulation experiments on randomly-generated workloads, to get a feel for typical (rather than worst-case) effectiveness — these are discussed in Section 6.2.1 below.

Theorem 1. *Let f_{opt} denote the minimum amount of computation that must be accommodated on an individual processor within each frame in any feasible m-processor CE for a given implicit-deadline periodic task system τ. Let $C_{\max}$ denote the largest WCET of any task in τ: $C_{\max} \overset{def}{=} \max_{\tau_i \in \tau}\{C_i\}$. The polynomial-time approximation algorithm of Section 6.1 above will successfully construct a CE for τ upon m processors, with each processor needing to accommodate no more than $(f_{opt} + C_{\max})$ amount of execution during any frame.*

Proof. Since (as we had argued in Section 4) an integer solution to the ILP represents an optimal CE, observe that the minimum value of f computed in an integer solution to an ILP would be equal to f_{opt}. And since the ILP is more constrained than the Linear Program, the minimum value for f computed in the (not necessary integral) solution to the LP obtained by the polynomial-time algorithm of Section 6.1 is $\le f_{\text{opt}}$. Let f_{LP} denote this minimum value of f computed as a solution to the LP; we thus have that $f_{\text{LP}} \le f_{\text{opt}}$.

In constructing the CE above, the polynomial-time algorithm of Section 6.1 schedules each job according to one of two rules:

1. If variable $x_{i_o j_o k_o}$ is assigned a value one in the solution to the LP, then job j_{i_o} is scheduled upon the j_o'th processor during frame Φ_{k_o}.
2. Any job j_{i_o} not scheduled as above is scheduled upon the processor-frame pair to which it gets matched in the bipartite matching.

Clearly, the jobs assigned according to the first rule would fit upon the processors if each had a computing capacity of f_{LP} within each frame. Now, observe that the matching in the bipartite graph assigns at most one job to each processor during any given frame; therefore, the *additional* execution assigned to any processor during any frame is $< C_{\max}$. Hence each processor could accommodate all the execution assigned it within each frame provided it had a computing capacity of at least $f_{\text{LP}} + C_{\max}$, which is $< (f_{\text{opt}} + C_{\max})$. □

The *speedup factor* of an algorithm A is defined to be smallest positive real number x such that any task system that is successfully scheduled upon a particular platform by an optimal algorithm is successfully scheduled by algorithm A upon a platform in which the speed or computing capacity of all processors are scaled up by a factor $(1+x)$.

Corollary 1. *The polynomial-time approximation algorithm of Section 6.1 has a speedup bound no larger than* 2.

Proof. By Theorem 1 above, if a CE can be constructed for task system τ by an optimal algorithm upon m speed-f_{opt} processors, it can be scheduled by the polynomial-time algorithm of Section 6.1 upon m speed-$\left(f_{\text{opt}} + C_{\max}\right)$ processors. The corollary follows from the observation that $C_{\max}$ is necessarily $\le f_{\text{opt}}$; hence $\left(f_{\text{opt}} + C_{\max}\right)/f_{\text{opt}}$ is $\le 2f_{\text{opt}}/f_{\text{opt}} \le 2$. □

6.2.1. *Experimental evaluation*

We saw above (Corollary 1) that the polynomial-time approximation algorithm of Section 6.1 has a speedup factor no worse than 2. We have conducted some experiments on randomly-generated synthetic workloads to further compare the performance of the approximation algorithm with the exact approach of solving the ILP.

Workload generation. The task system parameters for each experiment were randomly generated using a variant of the methods used in prior research such as [24,25,20], in the following manner:

- Task utilizations (U_i) were generated using the UUniFast algorithm [26].
- Task periods were set to be at one of $F \times \{1, 2, 3, 4\}$ (the frame size F was set equal to 25 ms in these experiments, in accordance with prior recent work on cyclic executives such as [24,25,20]). Periods were assigned randomly and uniformly over these four values. (Since we are restricting attention in this paper to implicit-deadline systems, job deadlines were set equal to their periods.)
- Task WCETs were determined as the product of utilization and period.
- All task systems in which one or more tasks had a WCET greater than minor cycle duration F, were discarded (since such systems are guaranteed to have no feasible non-preemptive schedules).
  (For some of our experiments, we needed task systems in which the largest WCET of any task (the parameter $C_{\max}$ of Theorem 1) was bounded at one-half of three-quarters the frame size. In generating task systems for these experiments, we discarded all task systems in which some task had WCET greater than the bound.)
- All the experiments assumed a four-processor platform ($m \leftarrow 4$).

Experiments conducted, and observations made. We conducted two sets of experiments; in each experiment within each set,

1. A task system was generated using the procedure detailed above, with a specified number of tasks, a specified total utilization, and for some experiments, a specified bound on $C_{\max}$.
   Each task system so generated was scheduled in two different ways.
2. First, it was scheduled non-preemptively by generating a linear program as described in Section 4.2, and then solved as an ILP using the Gurobi [15] optimization tool (instrumented to time out after two seconds of execution, earlier experiments indicating that for systems of 20 tasks on 4 processors, longer runs never improved upon the value obtained within the first two seconds).
3. Second, it was scheduled preemptively by solving the linear program obtained above as an LP (i.e., without any integrality constraints) using Gurobi, and then applying the technique described in Section 6.1 to obtain a non-preemptive cyclic executive. The maximum amount of computation assigned to any processor within an individual frame in this schedule was determined, and designated as $f_{\max}$.
4. The speedup factor needed by the polynomial-time approximation algorithm for this particular task system was then computed as
   $$\max\left(1, \frac{f_{\max}}{F}\right)$$
   (Recall that F denotes the frame size, chosen to equal 25 ms in our experiments.)

We now describe the two sets of experiments separately.

§1: Variation of speedup factor with system utilization. As explained above, the speedup bound of 2 identified in Corollary 1 above is a worst-case one. In this set of experiments, we set out to determine how the speedup factor of a randomly-generated system tends to depend upon the cumulative utilization of the task system. We therefore generated 400 task

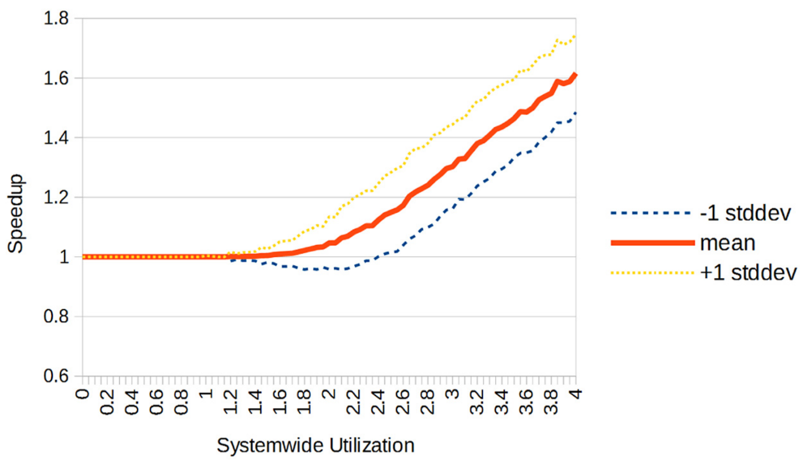

Fig. 5. Investigating how speedup factor changes with overall system utilization. The mean observed speedup factor over 400 task systems at each utilization is depicted, as is the range within one standard deviation from the mean.

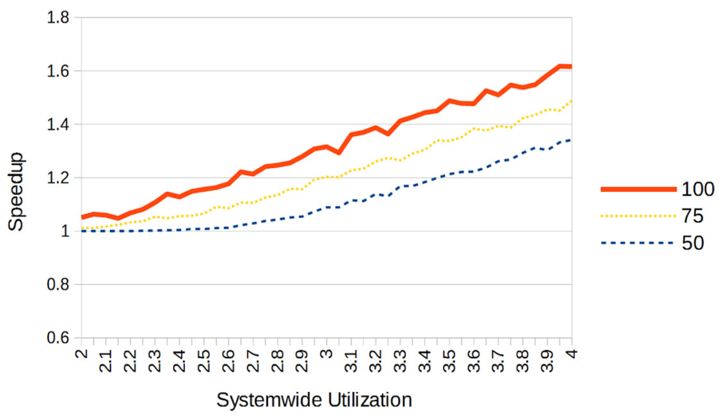

Fig. 6. Investigating how observed speedup factor depends upon $C_{\max}$, the largest WCET of any task. The mean observed speedup factor over 100 task systems is plotted, for $C_{\max}$ bounded at $\frac{1}{2}F$, $\frac{3}{4}F$, and F, where F denotes the frame size.

systems, each comprising 20 tasks, to have cumulative system utilization equal to U, for each value of U between 0 and 4 in steps of 0.05. The observed speedup factor needed by the approximation algorithm to schedule each task system was determined as described above, and the average and standard deviations computed. These values, plotted in Fig. 5, show a clear increasing trend: as overall utilization increases, so does the speedup factor needed to construct a non-preemptive schedule using the approximation algorithm.

§2: Variation of speedup factor with $C_{\max}$. Theorem 1 reveals that the speedup factor depends upon the value of $C_{\max}$, the largest WCET of any individual task. To investigate this relationship, we generated 100 task systems with overall utilization U for each value of U between 2 and 4 in steps of 0.05, in which the value of $C_{\max}$ was bounded from above at half the frame size, three quarters the frame size, and the full frame size. The observed speedup factor needed by the approximation algorithm to schedule each task system was determined as described above, and the average over the 100 individual task systems at each data point computed. These values, plotted in Fig. 6, show a clear increasing trend within each system utilization: the larger the bound on $C_{\max}$, the greater the observed speedup factor.

6.3. Special case: harmonic task systems

Let us now consider systems in which the tasks have harmonic periods: for any pair of tasks τ_i and τ_j, it is the case that T_i divides T_j exactly or T_j divides T_i exactly. Many highly safety-critical systems are explicitly designed to respect this restriction; additionally, many systems that are not harmonic are often representable as the union of a few — two or three — harmonic sub-systems.

For any job j_i, let us define $\mathcal{F}_i$ to be the set of frames that lie within j_i's scheduling window. For the task system of Example 1 (as depicted in Fig. 2), e.g., we have

$$\mathcal{F}_1 = \{\Phi_1, \Phi_2\}, \mathcal{F}_2 = \{\Phi_3, \Phi_4\}, \mathcal{F}_3 = \{\Phi_5, \Phi_6\}, \mathcal{F}_4 = \{\Phi_1, \Phi_2, \Phi_3\},$$

$\mathcal{F}_5 = \{\Phi_4, \Phi_5, \Phi_6\}$, and $\mathcal{F}_6 = \{\Phi_1, \Phi_2, \Phi_3, \Phi_4, \Phi_5, \Phi_6\}$.

Lemma 2. *For any two jobs j_i and j_j in harmonic task systems, it is the case that*

$$\left(\mathcal{F}_i \subseteq \mathcal{F}_j\right) \text{ or } \left(\mathcal{F}_j \subseteq \mathcal{F}_i\right) \text{ or } \left(\mathcal{F}_i \bigcap \mathcal{F}_j \text{ is empty}\right) \qquad \square$$

A polynomial-time approximation scheme (PTAS) was derived in [27] for the problem of *scheduling on restricted identical machines with nested processing set restrictions*; this PTAS can be directly applied to our problem of constructing non-preemptive cyclic executives for implicit-deadline periodic task systems with harmonic periods. This allows us to conclude that for the special case of harmonic task systems, polynomial-time approximation algorithms may be devised for constructing cyclic schedules that are accurate to any desired degree of accuracy.

7. Conclusions

Cyclic executives (CEs) are widely used in safety-critical systems industries, particularly in those application domains that are subject to statutory certification requirements. In our experience, current approaches to the construction of CEs are either ad hoc and based on the expertise and experience of individual system integrators, or make use of tools that are based on model checking or heuristic search.

Recent significant advances in the state of the art in the development of linear programming tools, as epitomized in the Gurobi optimizer [15], have motivated us to consider the use of linear programming for constructing CEs. We have shown that CEs for workloads that may be modeled as collections of implicit-deadline periodic tasks are easily and conveniently represented as linear programs (LPs). These LPs are solved very efficiently in polynomial time by LP tools like Gurobi; such solutions directly lead to preemptive CEs. If a non-preemptive CE is desired then one must solve an *integer* LP (ILP), which is a somewhat less tractable problem than solving LPs. However, our experiments indicate that Gurobi is able to solve most ILP problems representing non-preemptive CEs for collections of implicit-deadline periodic tasks quite effectively in a reasonable amount of time. We have also developed an approximation algorithm for constructing non-preemptive CEs that runs in polynomial time, and performs quite favorably in comparison to the exact algorithm in terms of both a worst-case quantitative metric (speedup factor) and in experiments on randomly-generated synthetic workloads.

Acknowledgements

This research has been supported in parts by NSF grants CNS 1409175 and CPS 1446631, AFOSR grant FA9550-14-1-0161, and ARO grant W911NF-14-1-0499.

References

[1] M. Stigge, W. Yi, Models for real-time workload: a survey, in: Proceedings of a Conference Organized in Celebration of Professor Alan Burns' Sixtieth Birthday, 2013, p. 133.
[2] M. Stigge, Real-Time Workload Models: Expressiveness vs. Analysis Efficiency, Ph.D. Thesis, Uppsala University, 2014.
[3] T.P. Baker, A. Shaw, The cyclic executive model and Ada, in: Proceedings of the IEEE Real-Time Systems Symposium, 1988, pp. 120–129.
[4] T.P. Baker, A. Shaw, The cyclic executive model and Ada, Real-Time Syst. 1 (1) (1989) 7–25.
[5] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.
[6] J.W.S. Liu, Real-Time Systems, Prentice–Hall, Inc., Upper Saddle River, New Jersey, 2000, 07458.
[7] J.-M. André, A. Kung, P. Robin, Ox: Ada cyclic executive for embedded applications, in: T.-D. Guyenne (Ed.), On-Board Real-Time Software, Proceedings of the International Symposium held, ISOBRTS, 13–15 November, 1995, in: European Space Agency Special Publication, vol. 375, ESTEC, Noordwijk, the Netherlands, 1996, pp. 241–245.
[8] K. Schild, J. Würtz, Off-line scheduling of a real-time system, in: K.M. George (Ed.), Proceedings of the 1998 ACM Symposium on Applied Computing, SAC98, ACM Press, Atlanta, Georgia, 1998, pp. 29–38.
[9] J. Yépez, J. Guardia, M. Velasco, J. Ayza, R. Castañé, P. Martí, J.M. Fuertes, Ciclic: a tool to generate feasible cyclic schedules, in: Proceedings of the IEEE International Workshop on Factory Communication Systems, WFCS, 2006.
[10] P. Pop, P. Eles, Z. Peng, Scheduling with optimized communication for time-triggered embedded systems, in: Proceedings of the Seventh International Workshop on Hardware/Software Codesign, 1999, CODES '99, 1999, pp. 178–182.
[11] A.P. Ravn, M. Schoeberl, Cyclic executive for safety-critical java on chip-multiprocessors, in: Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems, JTRES '10, ACM, New York, NY, USA, 2010, pp. 63–69.
[12] L. Khachiyan, A polynomial algorithm in linear programming, Dokl. Akad. Nauk SSSR 244 (1979) 1093–1096.
[13] N. Karmakar, A new polynomial-time algorithm for linear programming, Combinatorica 4 (1984) 373–395.
[14] R. McNaughton, Scheduling with deadlines and loss functions, Manag. Sci. 6 (1959) 1–12.
[15] Gurobi Optimization, Inc., Gurobi Optimizer Reference Manual, 2016.
[16] W. Horn, Some simple scheduling algorithms, Nav. Res. Logist. Q. 21 (1974) 177–185.
[17] S. Baruah, N. Cohen, G. Plaxton, D. Varvel, Proportionate progress: a notion of fairness in resource allocation, Algorithmica 15 (6) (1996) 600–625.
[18] L. Ford, D. Fulkerson, Flows in Networks, Princeton University Press, Princeton, NJ, 1962.
[19] J. Ullman, NP-complete scheduling problems, J. Comput. Syst. Sci. 10 (3) (1975) 384–393.
[20] T. Fleming, S. Baruah, A. Burns, Improving the schedulability of mixed criticality cyclic executives via limited task splitting, in: Proc. 24th International Conference on Real-Time Networks and Systems, 2016, pp. 277–286.
[21] J.-K. Lenstra, D. Shmoys, E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Math. Program. 46 (1990) 259–271.

[22] L. Ford, D. Fulkerson, Maximal flow through a network, Can. J. Math. 8 (2018) 399–404.
[23] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2009.
[24] T. Fleming, A. Burns, Extending mixed criticality scheduling, in: Proceedings of the International Workshop on Mixed Criticality Systems, WMC, 2013.
[25] A. Burns, T. Fleming, S. Baruah, Cyclic executives, multi-core platforms and mixed criticality applications, in: Proceedings of the 2015 27th EuroMicro Conference on Real-Time Systems, ECRTS '15, IEEE Computer Society Press, Lund (Sweden), 2015.
[26] E. Bini, G. Buttazzo, Measuring the performance of schedulability tests, Real-Time Syst. 30 (1–2) (2005) 129–154.
[27] L. Epstein, A. Levin, Scheduling with processing set restrictions: PTAS results for several variants, Int. J. Prod. Econ. 133 (2) (2011) 586–595.