Multi-core Cyclic Executives for Safety-Critical Systems

Calvin Deutschbein[1], Tom Fleming[2], Alan Burns[2], and Sanjoy Baruah[3](✉)

[1] The University of North Carolina at Chapel Hill, Chapel Hill, USA
[2] The University of York, York, UK
[3] Washington University in St. Louis, St. Louis, USA
baruah@wustl.edu

Abstract. 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. The implementation of cyclic executives upon multi-core platforms is considered. A Linear Programming (LP) based formulation is presented of the problem of constructing cyclic executives upon multiprocessors for a particular kind of recurrent real-time workload – collections of implicit-deadline periodic tasks. Techniques are described for solving the LP formulation under different kinds of restrictions in order to obtain preemptive and non-preemptive cyclic executives.

1 Introduction and Motivation

Real-time scheduling theory has made great advances over the past several decades. Despite these advances, interactions with industrial collaborators in highly safety-critical application domains, particularly those that are subject to stringent certification requirements, reveal that the use of the very simple *cyclic executive* approach [1] remains surprisingly wide-spread for scheduling safety-critical systems. A cyclic executive (CE) is a simple deterministic scheme that consists, for a single processor, of the repeated execution of a series of *frames*, each comprising a sequence of *jobs* that execute in their defining sequence and must complete by the end of the frame. Although there are a number of drawbacks to using cyclic executives (some are discussed in Sect. 2), this approach offers two significant advantages, *predictability* and *low run-time overhead*, that are responsible for their continued widespread use in scheduling highly safety-critical systems.

Highly safety-critical real-time systems have traditionally been implemented upon custom-built single-core processors that are designed to guarantee predictable timing behavior during run-time. As safety-critical software has become more computation-intensive, however, it has proved too expensive to custom-build 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

K.G. Larsen et al. (Eds.): SETTA 2017, LNCS 10606, pp. 94–109, 2017.
https://doi.org/10.1007/978-3-319-69483-2_6

described here into the construction of CEs that are suitable for execution upon multi-core processors.

This research. We derive several approaches to constructing cyclic executives for implicit-deadline periodic task systems upon identical multiprocessors. These approaches share the commonality that they are all 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 [8], and hence unlikely to be solvable exactly in polynomial time. 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. 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.

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 [11, Chap. 5.2–5.4].

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

[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.

one or more *frames* (also known as minor schedules or minor cycles). 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. This 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). CE's are typically used for periodic workloads. Hence the schedule-generation approach proposed in [1] 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 [1]; however, heuristics have been developed that permit system developers to construct such schedules for reasonably complex systems (as Baker & Shaw have observed [1], "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.

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),

[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

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 CE's we will construct.

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: $j_i \stackrel{\text{def}}{=} (a_i, c_i, d_i)$. We refer to the interval $[a_i, d_i)$ as the *scheduling window* of this job j_i.) 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 believe 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.

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, and major cycle duration is 12.) For this τ, $\mathcal{J}$ comprises the six jobs j_1–j_6 depicted in Fig. 2. 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 with a brief review of some well-known facts concerning linear programs that we will use in later sections of the paper.

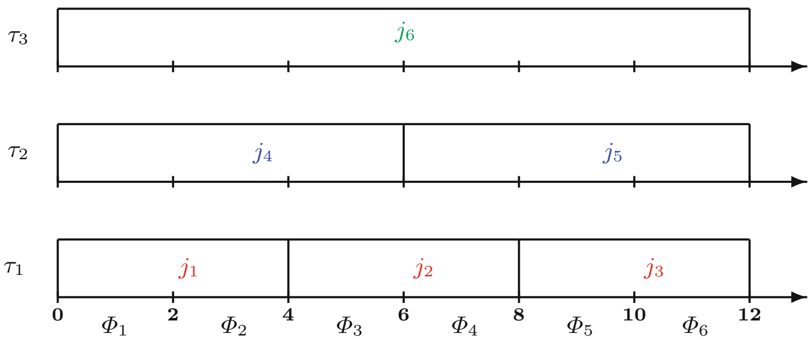

Fig. 2. The jobs generated by the task system of Example 1.

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 $\lambda, 0 \leq \lambda \leq 1$, 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 [9] or the interior point algorithm [7].

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 v 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 $\big(N \times m \times (P/F)\big)$ variables, each of which is subject to a non-negativity constraint (i.e., each may only take on a value ≥ 0), and $\big(n + (m+N) \times (P/F)\big)$ additional linear constraints.

4.2.1 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 $[1, m]$.
- 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 $[a_i, d_i)$.

The total number of x_{ijk} variables is equal to $\big(N \times m \times (P/F)\big)$, where N denotes the number of periodic tasks, m denotes the number of processors, and P/F represents the number of minor cycles.

4.2.2 An Objective Function

Let f denote the amount of computing that can be accomplished by a processor executing for the duration F of an entire frame; for unit-speed processors, $f = F$. We will define the following objective function for our LP:

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

The value of f obtained by solving the LP represents the minimum amount of computation needed to be completed by an individual processor within a duration F; if the available processors can indeed accommodate this amount of computation, then the solution is a feasible one.

4.2.3 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, these variables 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.

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

4.2.4 Solving the LP

With regards to the LP constructed above, 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 Sect. 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 Sects. 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, $1 \leq k_o \leq (P/F)$.

For each job j_{i_o} observe that

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

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 $\chi_{i_o} \leq f$ 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

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

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 [12] 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 $\chi_{i_o} \leq f$, 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 Sect. 6.2 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 [6]; performance of the Gurobi Optimization tool scaled very well with the size of the task system in these experiments.

6 Non-preemptive Cyclic Executives

We now discuss the process of obtaining 0/1 integer solutions to the linear program defined in Sect. 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.

Let us now take a closer look at the LP that was constructed in Sect. 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

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

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 $x_{i_o j k_o}$'s can equal 1, from which it follows that Constraint 4 necessarily holds for job j_{i_o} within 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

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

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 CE's 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 Sect. 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 $0 \leq x_{ijk} \leq 1$ for all variables x_{ijk}), and then obtain a basic solution[3] to the resulting linear program (without the constraint that variables take on integer values). As stated in Fact 2 of Sect. 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 Sect. 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 [10].

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 [10] 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 standard network-flow algorithms.

[3] Recall from Sect. 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.

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.

6.2 Evaluating the Approximation Algorithm

We now compare the effectiveness of the polynomial-time approximation algorithm of Sect. 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 te 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 Sect. 6.2 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: $C_{\max} \stackrel{def}{=} \max_{\tau_i \in \tau}\{C_i\}$. The polynomial-time approximation algorithm of Sect. 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 Sect. 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 $f_{\rm opt}$. 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 Sect. 6.1 is $\le f_{\rm opt}$. Let $f_{\rm LP}$ denote this minimum value of f computed as a solution to the LP; we thus have that $f_{\rm LP} \le f_{\rm opt}$.

In constructing the CE above, the polynomial-time algorithm of Sect. 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 $f_{\rm LP}$ 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 $f_{\rm LP} + C_{\max}$, which is $< (f_{\rm opt} + C_{\max})$. $\square$

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 Sect. 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 Sect. 6.1 upon m speed-$\Big(f_{\text{opt}} + C_{\max}\Big)$ processors. The corollary follows from the observation that $C_{\max}$ is necessarily $\leq f_{\text{opt}}$; hence $\Big(f_{\text{opt}} + C_{\max}\Big)/f_{\text{opt}}$ is $\leq 2f_{\text{opt}}/f_{\text{opt}} \leq 2$. □

Experimental Evaluation. We saw above (Corollary 1) that the polynomial-time approximation algorithm of Sect. 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 [3,5], in the following manner:

- Task utilizations (U_i) were generated using the UUniFast algorithm [2].
- Task periods were set to be at one of $F \times \{1, 2, 3, 4\}$ (the frame size F was set equal to 25ms in these experiments, in accordance with prior recent work on cyclic executives such as [3,5]). 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 Sect. 4.2, and then solved as an ILP using the Gurobi [6] 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 Sect. 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.

6.2.1 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 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. 3, 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.

6.2.2 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

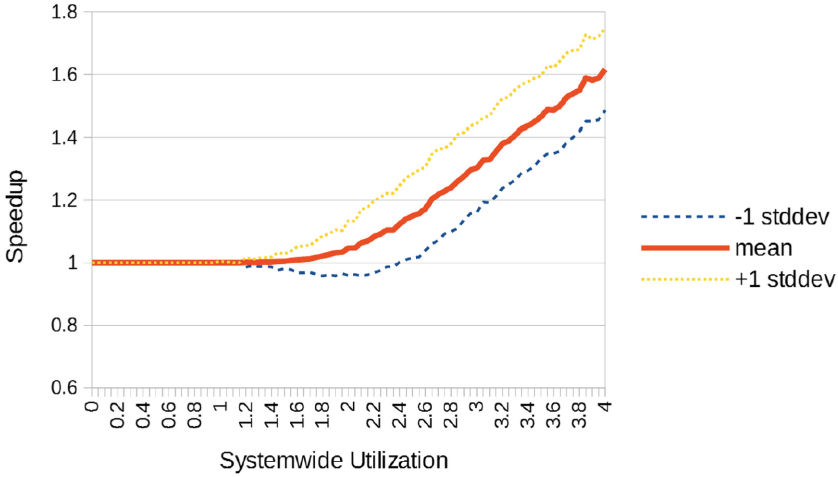

Fig. 3. Investigating how speedup factor changes with overall system utilization. The mean observed speedup factor over 400 task systems at each utilization is depicted, as is the range within one standard deviation from the mean.

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. 4, show a clear increasing trend within each system utilization: the larger the bound on $C_{\max}$, the greater the observed speedup factor.

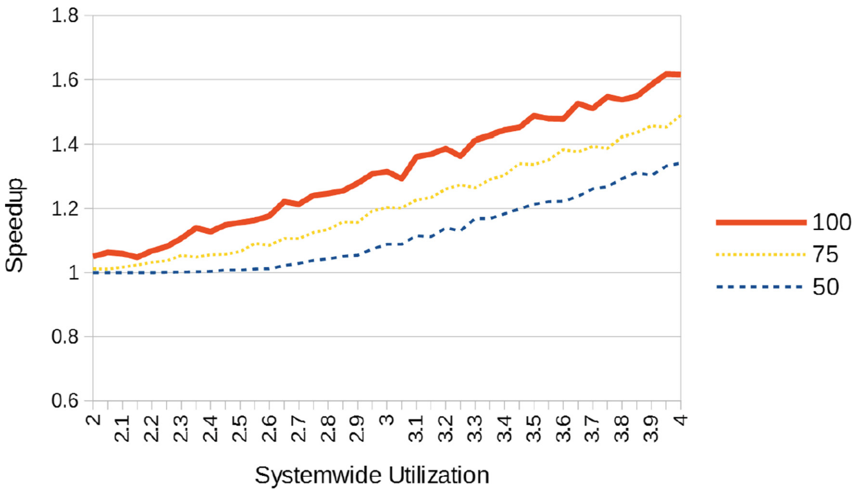

Fig. 4. Investigating how observed speedup factor depends upon $C_{\max}$, the largest WCET of any task. The mean observed speedup factor over 100 task systems is plotted, for $C_{\max}$ bounded at $\frac{1}{2}$, $\frac{3}{4}$, and 1 times the frame size.

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

$$\mathcal{F}_1 = \{\Phi_1, \Phi_2\}, \mathcal{F}_2 = \{\Phi_3, \Phi_4\}, \mathcal{F}_3 = \{\Phi_5, \Phi_6\}, \mathcal{F}_4 = \{\Phi_1, \Phi_2, \Phi_3\}, \mathcal{F}_5 = \{\Phi_4, \Phi_5, \Phi_6\},$$

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

Lemma 1: *For any two jobs j_i and j_ℓ in harmonic task systems, it is the case that*

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

A polynomial-time approximation scheme (PTAS) was derived in [4] 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 [6], 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 is supported by NSF grants CNS 1409175 and CPS 1446631, AFOSR grant FA9550-14-1-0161, and ARO grant W911NF-14-1-0499.

References

1. Baker, T.P., Shaw, A.: The cyclic executive model and Ada. In: Proceedings of the IEEE Real-Time Systems Symposium, pp. 120–129 (1988)
2. Bini, E., Buttazzo, G.: Measuring the performance of schedulability tests. Real-Time Syst. **30**(1–2), 129–154 (2005)
3. Burns, A., Fleming, T., Baruah, S.: Cyclic executives, multi-core platforms and mixed criticality applications. In: Proceedings of the 2015 27th EuroMicro Conference on Real-Time Systems, ECRTS 2015. IEEE Computer Society Press, Lund (Sweden) (2015)
4. Epstein, L., Levin, A.: Scheduling with processing set restrictions: PTAS results for several variants. Int. J. Prod. Econ. **133**(2), 586–595 (2011)
5. Fleming, T., Burns, A.: Extending mixed criticality scheduling. In: Proceedings of the International Workshop on Mixed Criticality Systems (WMC), December 2013
6. Gurobi Optimization Inc: Gurobi Optimizer Reference Manual (2016). http://www.gurobi.com
7. Karmakar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**, 373–395 (1984)
8. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
9. Khachiyan, L.: A polynomial algorithm in linear programming. Dokklady Akademiia Nauk SSSR **244**, 1093–1096 (1979)
10. Lenstra, J.K., Shmoys, D., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**, 259–271 (1990)
11. Liu, J.W.S.: Real-Time Systems. Prentice-Hall Inc., Upper Saddle River (2000)
12. McNaughton, R.: Scheduling with deadlines and loss functions. Manag. Sci. **6**, 1–12 (1959)