SOME COMPLEXITY RESULTS CONCERNING THE NON-PREEMPTIVE 'THRIFT' CYCLIC SCHEDULER

Michael Short
Embedded Systems Laboratory University of Leicester
University Road, Leicester, U.K.

Keywords: Embedded Systems, Non-preemptive scheduling, Feasibility Analysis, Complexity.

Abstract: Non-preemptive schedulers, despite their many perceived drawbacks, remain a very popular choice for practitioners of real-time and embedded systems. Although feasibility conditions for non-preemptive scheduling models have previously been considered in the literature, to date little attention has been paid to the non-preemptive 'thrift' (or 'TTC') cyclic scheduler. This type of scheduler differs from a standard 'cyclic executive' in that it does not allow the use of inserted idle-time, and it does not require a lookup table of task executions over the major cycle of the schedule; a feasible schedule is effectively created by assigning release times ('offsets') to the tasks. To this end, this paper seeks to address the complexity of generating a feasible cyclic schedule for such a scheduler. It will be shown that when a single set of release times is assigned to the tasks, deciding feasibility of the resulting schedule is coNP-Complete (in the strong sense); and the release time assignment problem for such a scheduler is complete for $\sum_2^P$.

1 INTRODUCTION

In many real-time embedded systems, some form of scheduler is generally used instead of a full "real-time operating system" to keep the software environment as simple as possible. In general, such systems may be designed around several basic paradigms: time-triggered or event-triggered, and preemptive or co-operative (non-preemptive). This paper is concerned with single-processor time-triggered, non-preemptive schedulers.

Such architectures have been found to be a good match for a wide range of low-cost, resource-constrained applications. These architectures also demonstrate comparatively low levels of task jitter, CPU overheads, memory resource requirements and power consumption (Pont 2001; Baker & Shaw 1989; Short et al. 2008; Burns et al. 1994). Additionally, such schedulers exhibit extremely high levels of predictability and determinism. Exploratory studies also seem to indicate better transient error recovery properties in these systems than their preemptive counterparts (Short et al. 2008).

Because of these properties, non-preemptive schedulers have proved to be extremely effective at implementing systems such as process controllers, robotics, automotive applications and other types of system in which reliability is a key design goal (e.g. Gendy & Pont 2008; Short et al. 2008; Burns et al. 1994; Pont 2001).

Although non-preemptive systems are inherently free of deadlocks and other concurrency issues by the very nature of their design (Pont 2001; Baker & Shaw 1989; Short et al. 2008), the fact that tasks cannot preempt one another introduces several complexity issues related to feasibility analysis (Garey & Johnson 1979). A feasible task set is one in which all jobs generated by all tasks in the system can be said to meet their deadlines over the lifetime of the system.

Various differing models for the implementation of a 'cyclic executive' – a basic time-triggered non-preemptive design - have been proposed over the years, along with appropriate discussion and mathematical techniques for feasibility analysis. Relatively recently, the non-preemptive 'thrift' cyclic scheduler (or simply TTC scheduler) has been proposed (Pont 2001). This scheduling algorithm essentially maintains a system of congruence's to achieve its behaviour, and is presented in a simplified form in Figure 1. As can be seen, the scheduler is driven by a periodic timer signal (typically an interrupt). When tasks are released by the scheduler, they are immediately dispatched on a first-come, first-served basis (FCFS).

Short M. (2009).
SOME COMPLEXITY RESULTS CONCERNING THE NON-PREEMPTIVE 'THRIFT' CYCLIC SCHEDULER.
In *Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics - Intelligent Control Systems and Optimization*, pages 347-350
DOI: 10.5220/0002177603470350

```
START
tick :=0; // Initialize the clock 'tick' variable
DO(FOREVER) // Enter infinite loop
   FOR i := 1 TO n DO // n is the number of tasks
      IF(((tick – ri) mod(pi)) = 0)
         Run(ti)// Immediately execute any released task
   END FOR
   Wait_Timer(); // Wait for the next timer tick to occur
   tick := tick + inc; // Increase the time index by a prefixed increment factor
END DO
EXIT
```

Figure 1: Thrift scheduling algorithm.

As can be seen, the scheduler does not support the use of 'inserted idle time', and thus does not require a lookup table of task and process executions to be created over the duration of the major cycle, which is a fundamental requirement in a 'standard' cyclic executive (Baker & Shaw 1989; Burns et al. 1994). This fundamental difference allows the scheduler to better handle tasks with larger and non-harmonic periods, requiring only storage space *O(n)* as compared to *O(nh)* where *n* is the number of tasks and *h* is the major cycle length.

In the case of the non-preemptive thrift scheduler, generating a feasible schedule surmounts to assigning specific release times to tasks such that a 'task overrun' – an overload of the processor in any given time quantum - does not occur (Pont 2001; Gendy & Pont 2008). The time quantum is normally referred to as the scheduler 'tick' interval t_{Tick} and is generally chosen to be as large as possible given the task periods, according to (1):

$$t_{Tick} = \gcd(p_1, \ldots, p_n) \tag{1}$$

With such an arrangement, w.l.o.g. the task release times can be specified as integer multiples of t_{tick} (Pont 2001; Gendy & Pont 2008). Since it has been suggested that creating the lookup table for a 'standard' cyclic executive is NP-Hard (Burns et a. 1994), this short paper seeks to explore the complexity of the offset assignment process in the 'thrift' cyclic scheduler. In Section 2 of the paper, a simple polynomial-time sufficient condition for feasibility is developed; however it is clear that this simple condition is very pessimistic. In Section 3 of the paper, the complexity of designing a TTC schedule is then considered, and it will be shown that even verifying the feasibility of a candidate solution is coNP-Complete (in the strong sense), i.e. intractable. This Section also shows that the problem of assigning specific release times to tasks is complete for the second level of the polynomial hierarchy, i.e. NP^{NP}-Complete using the notation of Garey & Johnson (1979). Section 4 concludes the paper.

2 A SIMPLE FEASIBILITY TEST

As mentioned in Section 1, the primary condition for feasibility in a non-preemptive thrift cyclic schedule is that task overruns do not occur. Given a set of tasks τ, where each $t_i \in \tau$ can be represented by a tuple:

$$t_i = (p_i, c_i, r_i) \tag{2}$$

In which p_i is the task period, c_i is the (worst case) computation time of the task and r_i is the task release time (note that a specific relative deadline can be omitted). A necessary condition for feasibility is that the computation time of each task is less than or equal to t_{tick}, and it follows that a very simple sufficient condition for feasibility of the task set is as follows:

$$\sum_{i \in \tau} c_i \leq t_{Tick} \tag{3}$$

It is quite straightforward to see that if the summed execution times of the tasks does not exceed the tick interval, then regardless of the choice of task release times (which are integer multiples of t_{tick}) then a task overrun cannot occur. It is also trivial to observe that this condition is very pessimistic, as shown by the example in Figure 2, with the tasks having periods and execution times given by {5, 2}, {15, 2 and {15,2}; resulting in a tick interval of 5 according to (1). This Figure also highlights the effect that the choice of task release times has on feasibility. In the following Section, the complexity of this release time assignment process will be investigated.

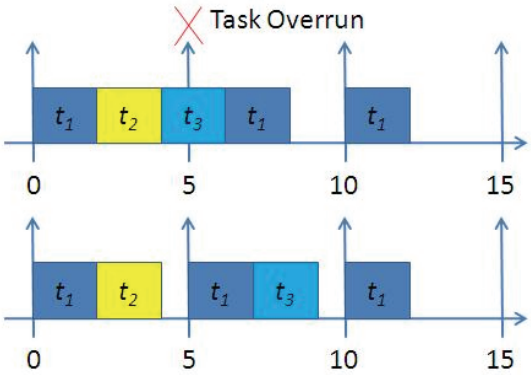

Figure 2: Effect of task release times on feasibility, showing (top): a synchronous infeasible task set; (bottom): an asynchronous feasible task set with release time of 1 tick added to t_3.

3 THE COMPLEXITY OF AN EXACT FEASIBILITY TEST

3.1 Solution Verification

Suppose we have been given a candidate solution (feasible schedule) for a thrift cyclic scheduler, i.e a set of tasks with specific release times; it is natural to consider the complexity of verifying that this solution is actually feasible. It will now be shown that deciding the feasibility of such a 'concrete' thrift schedule (the FTS problem) is strongly coNP-Complete. Membership of the problem in coNP follows since, form Figure 1, given a tick interval j in which a task overrun occurs, the execution time commanded in this interval may be determined in time proportional to n. Prior to giving the hardness proof, the simultaneous congruences problem (SCP) will be introduced; SCP was shown to be NP-Complete, in the strong sense, by Baruah et al. (1990):

Simultaneous Congruences Problem (SCP)

Instance: A set A of ordered integer pairs $\{(x_1, y_1) \ldots (x_n, y_n)\}$ and a positive integer $k > 1$.

Question: Is there a subset $A' \subseteq A$ of k ordered pairs, and a positive integer z, such that for all $(x_i, y_i) \in A'$, $z \equiv x_i \bmod(y_i)$?

Theorem 1: FTS is strongly coNP-Complete.

Proof: Transformation from the compliment of SCP. Let $\phi = < (x_1, y_1) \ldots (x_n, y_n), k >$ denote an arbitrary instance of SCP. From this a set of n concrete tasks for an instance of FTS are created according to (4), with a value of t_{tick} equal to k-1. This transformation can be performed in time proportional to n and is hence polynomial.

$$\begin{aligned} p_i &= x_i t_{tick} \\ c_i &= 1 \\ r_i &= y_i t_{tick} \end{aligned} \qquad (4)$$

Next, it is argued that a positive solution to ϕ exists iff there is a negative answer to FTS. If FTS is negative, then it implies that during at least one tick interval j, the processor demand is greater than t_{tick}. Since t_{tick} is equal to k-1, and given our construction of the task execution times, at least k tasks must be simultaneously released at the start of tick interval j; this gives a solution to ϕ with a certificate j. Conversely, if the answer to FTS is positive, then the peak processor demand is $\leq t_{tick}$ and a tick interval in which k or more tasks are simultaneously released does not exist; implying a negative answer to ϕ. Since SCP is strongly NP-Complete, the Theorem is proved. □

3.2 Release Time Assignment

Given this result, it is clear that if the verification of a candidate solution is coNP-Complete, then the problem of assigning release times (the 'TS' problem) is strongly coNP-Hard. However, since there also seems to be an exponential number of possible start times for a thrift scheduling instance, under the assumption that P ≠ NP it is also worthwhile investigating exactly where this problem lies on the so-called 'polynomial hierarchy' (Garey & Johnson 1979). It can be seen that the task release times for 'Yes' (feasible) instances of this problem can be encoded in a number of bits that is less than or equal to the task set periods, and hence the size of the overall TS problem instance. Given the previous Theorem, the resulting schedule is verifiable in polynomial time by a Turing machine with an oracle for the FTS problem; the problem resides in Σ_2^p. To show that the problem is complete for this complexity class, the Periodic Maintenance Scheduling Problem (PMSP) is now introduced. This problem is known to be Σ_2^p – Complete (Baruah et al. 1990).

PERIODIC MAINTENANCE SCHEDULING PROBLEM (PMSP)

Instance: A set C of ordered pairs $\{(n_1, c_1) \ldots (n_n, c_n)\}$, with each c_i representing a maintenance activity having an integer period n_i, positive integer $k > 1$.

Question: Is there a mapping of the activities in C to positive integer time slots such that successive occurrences of each c_i are exactly n_i time slots apart, and no more than k activities ever collide in a single slot?

Theorem 2: TS is Σ_2^p - Complete.

Proof: Transformation from PMSP.

Let $C = < ((c_1, n_1) \ldots (c_n, n_n), k >$ denote an arbitrary instance of PMSP. From this a set N of n tasks to be scheduled by TS are created according to (5), with a value of t_{tick} equal to k:

$$p_i = n_i t_{tick} \qquad c_i = 1 \tag{5}$$

Again this transformation can be performed in polynomial time. Next, it is argued that a solution to C exists iff N can be scheduled by TS. If the answer to this instance of TS is 'Yes', this implies that release times can be assigned to each task in N such that a task overrun does not occur, which from the transformation given implies that a maintenance schedule for C - in which no more than k activities occur simultaneously - also exists with a certificate $(r_1 \ldots r_n)$. Conversely, if the answer to TS is 'No', then a schedule in which a task overrun does not occur does not exist for any combination of task release times, implying that a maintenance schedule for C - in which no more than k activities occur simultaneously - does not exist. Since PMSP is Σ_2^p - Complete, the Theorem is proved. □

4 CONCLUSIONS

This paper has considered the complexity of generating a feasible cyclic schedule for the non-preemptive 'thrift' cyclic scheduler. It has been shown that an efficient - but also very pessimistic - sufficient feasibility condition exists. However, an exact solution to the problem requires the assignment of specific release times to the tasks; the complexity of this problem has been shown to be NP^{NP}-Complete, with the verification of a candidate solution being strongly coNP-Complete.

These results imply that heuristic techniques that may be used to solve the TS problem - such as the algorithm proposed by Gendy & Pont (2008) – must be strongly coNP-Hard if they provide any confirmation that the resulting schedule is indeed feasible. In conclusion, although the thrift cyclic scheduler possesses many desirable qualities, extreme computational difficulties may occur when designing a schedule. In such cases, then the non-preemptive EDF scheduling algorithm (shown to be optimal among the non-idling non-preemptive scheduling strategies by Jeffey et al. 1991) may be beneficial.

REFERENCES

Baker, T.P. and Shaw, A., 1989. The cyclic executive model and Ada, Real-Time Systems, Vol. 1, No. 1, pp. 7-25.

Baruah, S.K., Rosier, L.E. and Howell, R.R., 1990. Algorithms and Complexity concerning the preemptive scheduling of periodic tasks on one processor, Real-Time Systems, Vol. 2, No. 4, pp. 301-324.

Burns, A., Hayes, N. and Richardson, M.F., 1995. "Generating Feasible Cyclic Schedules", Control Engineering Practice, Vol. 3, No. 2, pp. 151-162.

Garey, M.R. and Johnson, D.S., 1979. Computers and Intractability: A guide to the Theory of NP-Completeness, W.H. Freeman & Co Ltd, April 1979.

Gendy, A.K. and Pont, M.J., 2008. Automatically configuring time-triggered schedulers for use with resource-constrained, single-processor embedded systems, IEEE Trans. on Industrial Informatics, Vol. 4, No. 1, pp. 37-45.

Jeffay, K., Stanat, D.F. and Martel, C.U., 1991. On non-preemptive scheduling of periodic and sporadic tasks, In Proceedings of the 12th IEEE Symposium on Real-Time Systems, pp. 129-139.

Pont, M.J., 2001. Patterns For Time Triggered Embedded Systems, ACM Press / Addison Wesley.

Short, M., Pont, M.J. and Fang, J., 2008. Exploring the impact of preemption on dependability in time-triggered embedded systems: A pilot study, In: Proceedings of the 20th Euromicro conference on real-time systems (ECRTS 2008), Prague, Czech Republic., pp. 83-91, 2-4 July 2008.