Scheduling of Offset Free Systems

Author: Joél Goossens

Presenters: Ilge Akkaya
Forrest Iandola

Introduction

- Correctness of real-time computations depend on
  - logical/computational results AND
  - when the result is made available
- Predictability (whether the system meets all its timing requirements or not) is therefore an important property of real-time systems
- **Hard** deadlines : missing one leads to system failure
- **Soft** deadlines : missing one can be tolerated

Scheduling Periodic Hard Real-Time Tasks: Definitions

- Single-Processor implementation
- Each Task $\tau_i = (T_i, D_i, C_i, O_i)$
  - T_i : task period
  - D_i : Hard deadline delay
  - C_i : Execution Requirement (WCET(τ_i))
  - O_i : Offset
  - ($0 < C_i < T_i$)
- The requests of τ_i occur at O_i+(k-1) T_i (k=1,2,…)
- Tasks must finish by O_i+(k-1) T_i+D_i

Classification of Periodic Task Sets - Offsets

- **Synchronous**.
  - Fixed-offset ($O_1=O_2=\ldots=O_i$)
- **Asynchronous**.
  - Arbitrary but pre-defined offsets for each periodic task
- **Offset Free**.
  - No definite requirement about task start times
  - Scheduler chooses offsets

Classification of Periodic Task Sets - Deadlines

- **Implicit deadline**
  - Deadline equals period
  - i.e. each request must be completed before the next request
- **Constrained deadline**.
  - $D_i <= T_i$
- **Arbitrary deadline**.
  - Deadline may be less or greater than the period
  - Many incomplete requests of the same task may exist simultaneously

Scheduler Algorithms

- **Dynamic**
  - Task priorities computed during execution
    - Deadline-Driven (Earliest Deadline First (EDF))
    - Least Laxity First (assigns priority based on the amount of "slack time")
    - Both optimal for all task sets considered
- **Static**
  - Task priorities are known a priori
  - Highest priority request executed first in runtime
  - Well-known static scheduling algorithms include:
    - Rate-Monotonic Scheduling (optimal for synchronous implicit deadline)
    - Deadline-Monotonic Scheduling (optimal for synchronous constrained deadline)

Static Scheduler Algorithms

- **Synchronous case** is the most limiting:
- The system being schedulable in synchronous tasks case **implies** system is also schedulable in all asynchronous cases
- Liu and Layland (Liu et al. 1973) show that **worst-case response time** of a **synchronous** task τ_i happens for its **first** request. (holds for implicit and constraint deadline systems)
- For arbitrary deadline systems, largest response time still happens for synchronous case (but not necessarily for the first request)

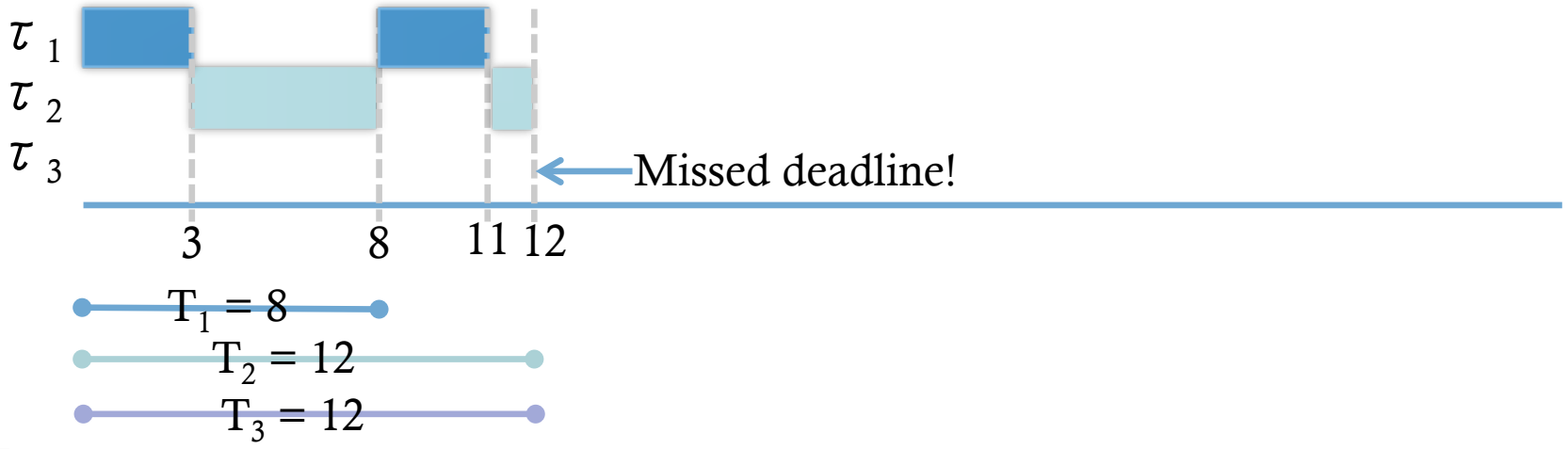
Static Scheduler Algorithms

- **Synchronous case** is the most limiting:
- The system being schedulable in synchronous tasks case **implies** system is also schedulable in all asynchronous cases
- Liu and Layland (Liu et al. 1973) show that **worst-case response time** of a **synchronous** task happens for its **first** request. (holds for implicit and constraint deadline systems) τ_i
- For arbitrary deadline systems, largest response time still happens for synchronous case (but not necessarily for the first request)
- Bottom line: **synchronous** case is usually pessimistic.

Example: Synchronous Case

Task	Period =Deadline	Priority	WCET
τ_1	8	1	3
τ_2	12	2	6
τ_3	12	3	1

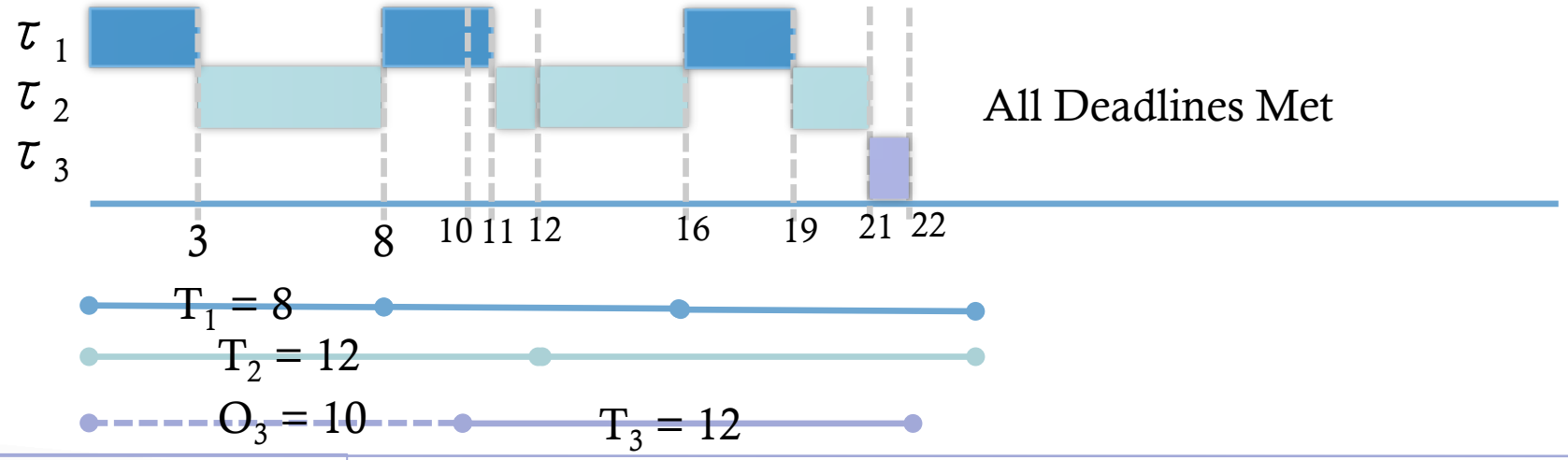
Synchronous Model



Example: Asynchronous Case

Task	Period =Deadline	Priority	WCET	Offset
τ_1	8	1	3	0
τ_2	12	2	6	0
τ_3	12	3	1	10

Asynchronous Model



Offset Free Systems

- If task set is unschedulable in the synchronous case, is there a set of offsets/priorities that make the system schedulable?
- Monotonic priority assignments are shown to be non-optimal for such systems
- If offsets can be chosen freely, possible to improve WCRT

Offset Granularity

- If task set is unschedulable for all non-negative integer offset assignments, is it also true it is also not schedulable for non-integer offset values?

> *Definition 1.* Let $S = \{\tau_i = \{T_i, C_i, D_i\} \mid i = 1, \ldots, n\}$ with $T_i, C_i, D_i \in \mathbb{N}$ be an offset free system. An offset assignment is said to have *a granularity of* m iff m is the smallest positive integer such that $O_i \in \{\frac{p}{m} \mid p \in \mathbb{N}\}$ $(i = 1, \ldots, n)$.

- Theorem 1: If S is not schedulable with a given **static priority assignment** and integer offsets, it is also not schedulable for all offset assignments with a granularity of m, for all m.
- Theorem 2: If S is not schedulable with an **EDF scheduler** for integer offset assignments, then it is also not schedulable for all offset assignments with granularity m.

Non-Equivalent Asynchronous Systems

- No specific scheduling algorithm or family (static/dynamic) considered
- Assume periodic schedule P:=lcm$\{T_i\}$ (P: hyper period)
- Periodic behavior only depends on the relative phasing of task requests
- Assuming a granularity of 1
- **Two asynchronous arbitrary deadline systems S and S' are equivalent,** if they have the same periods, deadlines, WCETs, and their offsets have the relation

$$O_i = O'_i + k_i T_i + A, \text{ where A is an integer.}$$

Systems Equivalent to Synchronous Case

- Theorem 3: Two equivalent asynchronous arbitrary deadline systems have the same periodic behavior.
- An asynchronous arbitrary deadline system S is said to be equivalent to its synchronous case, if there is a common t, such that for all tasks i, it satisfies $t = O_i + k_i T_i$
- Chinese Remainder Theorem => if periods of all tasks are relatively prime, the asynchronous system is always equivalent to the synchronous case. (with hyper-period $P = T_1 x T_2 x \ldots T_n$)

Non-equivalent asynchronous situations

- If task periods are not relatively prime…
- $O_i \equiv O_j \pmod{\gcd(T_i, T_j)} \qquad 1 \leq i < j \leq n$; ⇔ the asynchronous system is equivalent to the corresponding synchronous one

Generalized Chinese Remainder Theorem

- Still infinitely many asynchronous systems
- Theorem 6: Restrict offsets into classes such that

$$\begin{cases} O_1 = 0, \\ O_i \in [O_{i-1}, O_{i-1} + T_i) \qquad i = 2, \ldots, n. \end{cases}$$

- Number of classes of equivalent asynchronous systems is finite and is upper bounded by the product of task periods:

$$\prod_{i=2}^{n} T_i$$

THEOREM 9 *Let* $\Gamma = \{\tau_i = \{C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$. *There are* $\frac{\prod_{i=1}^{n} T_i}{P}$ *different equivalence classes of offset assignments.*

Optimal Scheduling of Offset Free Systems

- Previously shown that rate/deadline monotonic priority assignments are non-optimal
- For asynchronous systems, there exists an optimal static priority assignment that runs in $O(n^2)$ (Audsley, 1991)
- **Goal**: Extending this work, generalize results to offset free systems for **static** scheduling
- Among the **dynamic** schedules, EDF and laxity first are still optimal for asynchronous case.

Optimal Scheduling of Offset Free Systems

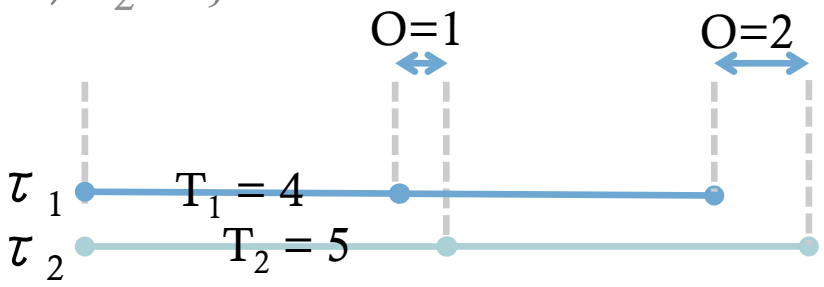
- Assume some scheduling rule Q (may be an optimal technique or a sub optimal one such as RM or DM)
- Choose the Q-optimal set of offsets
- An offset assignment rule A is Q-Optimal for an offset-free task family if,
  - When a feasible offset assignment B exists for a task set using Q,
  - The offset assignment A is also feasible for that task set with the scheduling rule Q
- Possible to do this exhaustively by considering Ti possible values of Oi and searching for the optimum. (Thm 6)

A Better Way…

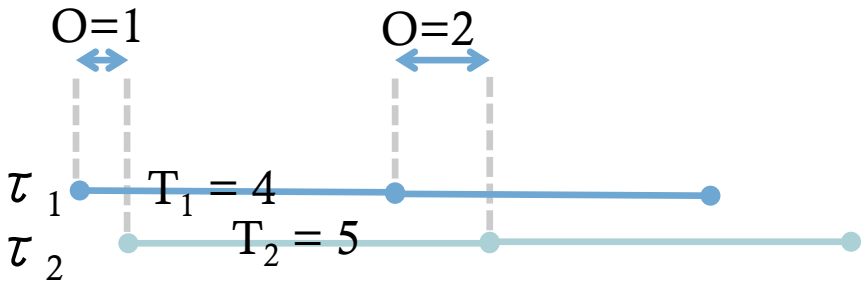
- Following Thm 9, only consider the $\frac{\prod_{i=1}^{n} T_i}{P}$ offset assignments.
- Two tasks case:
  - w.l.o.g, $O_1 = 0$. Goal is to find:
  - (T1xT2)/lcm{T1,T2} = gcd{T1,T2} choices of O_2.

A Better Way… Example:

- Task 1: $\{T_1=4, O_1=0\}$
- Task 2: $\{T_2=5, O_2=?\}$
- If $O_2=0$



- If $O_2=1$



Choices of $O_2=0$, $O_2=1$, $O_2=2$, are equivalent, they generate the same relative phasings between requests of task 1 and task2

A Better Way…

- Two choices O2 = O1+v1 and O2=O1+v2 are equivalent if they have the same relative phasing. i.e., if

$$v_1 \equiv v_2 \pmod{\gcd\{T_1, T_2\}}$$

In the case of n tasks…

- Fix O1=0
- Assume non-equivalent choices for O2
- Consider next the non-equivalent choices for O3, etc.
- For the i^{th} offset, there are $\gcd\{T_i, \text{lcm}\{T_1,\ldots,T_{i-1}\}$ choices.
- Theorem 10: This way, all non-equivalent asynchronous systems are yielded.

THEOREM 10

$$\prod_{i=2}^{n} \gcd\{T_i, \text{lcm}\{T_1, \ldots, T_{i-1}\}\} = \frac{\prod_{i=1}^{n} T_i}{\text{lcm}\{T_i | i = 1, \ldots, n\}}.$$

Dissimilar Offset Assignment

- So far showed that there are finitely many non-equivalent offset assignments to be considered: (product of periods)
- This can further be reduced down to $\frac{\prod_{i=1}^{n} T_i}{P}$ assignments
- Still exponential in n!
- Moving onto a heuristic algorithm that considers a single offset value for each task
- Runs in **polynomial** time and succeeds in scheduling most systems that are not schedulable in the synchronous case.

Points to Consider

- For various classes of periodic task sets, check if

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

- Check if the system is schedulable in the synchronous case (time complexity of synchronous case usually much simpler)
- If the system is not synchronously schedulable but U<=1, there could still be a valid offset assignment set, such that system becomes schedulable.
- **Idea: Move away from the synchronous (worst) case as much as possible**

Proximity to Synchronous Case

- Consider a minimal length L such that for any two tasks τ_1 and τ_2, there is an interval of length L that contains requests from both.
- Note that **L = 0** for the synchronous case (all requests arrive simultaneously at first)
- The larger L is, the more requests are dissimilar in the whole schedule
- For two tasks, the choice of offset that maximizes L is

THEOREM 11 *Let* $r \in [0, \gcd\{T_i, T_j\})$. *If* $O_i = O_j + r$ *(or* $O_j = O_i + r$*), the minimum distance between a request of* τ_i *and a request of* τ_j *is* $\min\{r, \gcd\{T_i, T_j\} - r\}$.

A heuristic Algorithm

- Pairwise offset assignments. (Oi, Oj)
- Case 1: Both offsets unassigned: randomize Oi, set Oj = Oi + floor(gcd(Ti,Tj)/2)
- Case 2: Set Oj only
- Case 3: Move onto the next pair
- No longer Q-optimal, but not NP-complex either.

Algorithm

Algorithm 1 The dissimilar offset assignment

1: $G \Leftarrow \{(i, j, \gcd(T_i, T_j)) \mid 1 \leq i < j \leq n\}$;
2: $\mathcal{G} \Leftarrow ((i_1, j_1, \gcd(T_{i_1}, T_{j_1})), (i_2, j_2, \gcd(T_{i_2}, T_{j_2})), \ldots)$, with $\{(i_k, j_k, \gcd(T_{i_k}, T_{j-k}) \mid k = 1, \ldots, \frac{n(n-1)}{2}\} = G$, such that $r < p \Longrightarrow \gcd(T_{i_r}, T_{j_r}) \geq \gcd(T_{i_p}, T_{j_p})$;
3: {The vector $\mathcal{G}$ is a sorted version of the set G. In the following we shall use the "dot notation" to denote the 3 fields of each entry of $\mathcal{G}$, which are *row*, *col* and *gcd*, respectively.}
4: $assignment \Leftarrow n$; {The remaining number of offset assignments.}
5: $Mark \leftarrow$ (false, ... ,false); {n components.}
6: $k \Leftarrow 1$;
7: **while** $assignment > 0$ **do**
8: **if** $\neg(Mark_{\mathcal{G}_k.col}) \wedge \neg(Mark_{\mathcal{G}_k.row})$ **then**
9: $O_{\mathcal{G}_k.row} \Leftarrow \text{rand}(); O_{\mathcal{G}_k.col} \Leftarrow O_{\mathcal{G}_k.row} + \mathcal{G}_k.gcd \text{ div } 2;$
10: $assignment \Leftarrow assignment - 2; Mark_{\mathcal{G}_k.row} = \text{true}; Mark_{\mathcal{G}_k.col} = \text{true};$
11: **else if** $\neg(Mark_{\mathcal{G}_k.col})$ **then**
12: $O_{\mathcal{G}_k.col} \Leftarrow O_{\mathcal{G}_k.row} + \mathcal{G}_k.gcd \text{ div } 2;$
13: $assignment \Leftarrow assignment - 1; Mark_{\mathcal{G}_k.col} = \text{true};$
14: **else if** $\neg(Mark_{\mathcal{G}_k.row})$ **then**
15: $O_{\mathcal{G}_k.row} \Leftarrow O_{\mathcal{G}_k.col} + \mathcal{G}_k.gcd \text{ div } 2;$
16: $assignment \Leftarrow assignment - 1; Mark_{\mathcal{G}_k.row} = \text{true};$
17: **end if**
18: $k \Leftarrow k + 1;$
19: **end while**

- Subtitles: Assign O_2 to task2 such that it maximizes the minimum distance between two requests of task1 and task2.
- Algorithm fixes n offsets of the periodic task set by considering the values gcd{Ti,Tj}, by decreasing order
- Avoid synchronization with the already chosen offsets (worst-case!)
- Works with any scheduling algorithm

Complexity

- The worst-case time complexity of the presented algorithm is $O(n^2(\log(T_{max})+\log n^2))$ and the maximal space complexity is $O(n^2)$
- Experimental Results: Algorithm Leads to 82% optimal scheduling
- Tested for n in [5,13] and Ti's in [5,30], U in [.65,1]
- Synchronous case turns out to be a loose upper bound for the feasibility – heuristic methods can do much better

Improvements

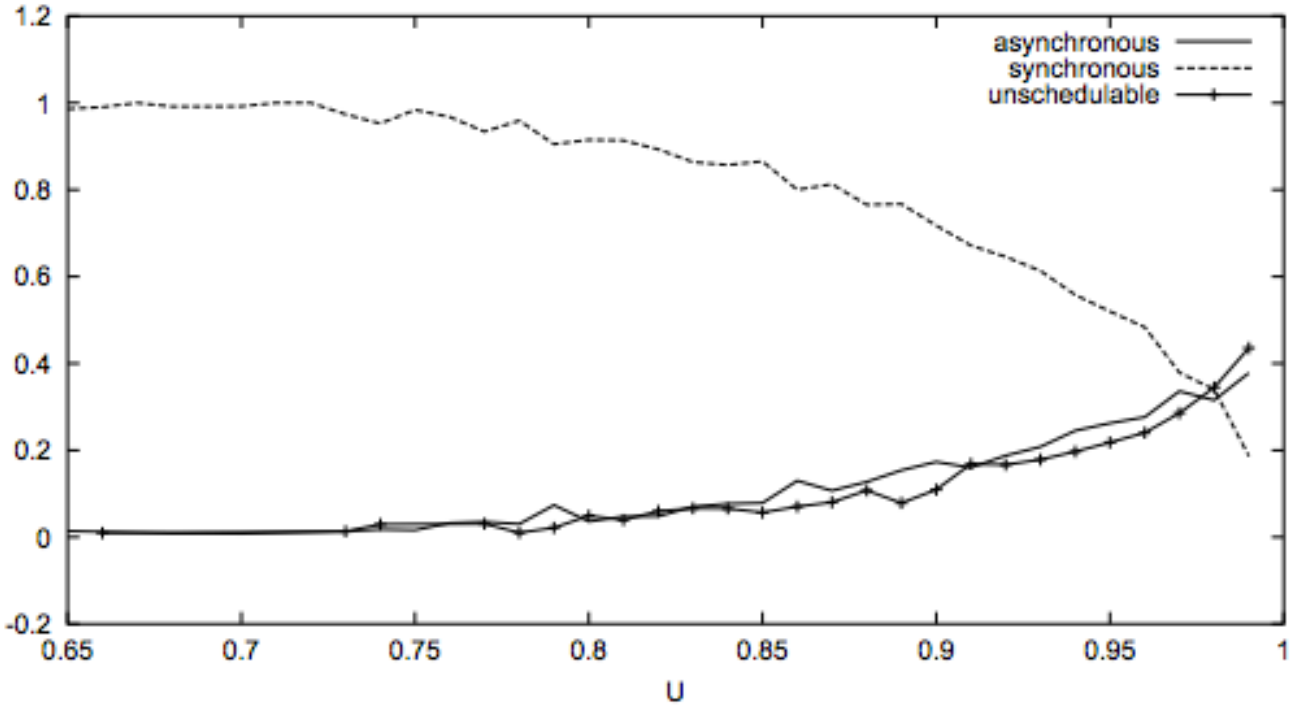


Figure 7. The proportion of systems schedulable in the synchronous case, schedulable only in an asynchronous situation and unschedulable whatever the offsets, in function of U.

Remaining Challenges

- Optimal static priority assignments for offset free systems (pseudo optimal, heuristic)
- Investigation of other heuristics based on Ci's and Di's