Mixed-Criticality Systems

Real-Time Industrial Systems

Marcello Cinque

Roadmap

- Mixed criticality Systems
- Vestal's model
- Hierarchical Scheduling

- References:
  - A. Burns and R. I. Davis. "Mixed Criticality Systems: a review"
  - S. Vestal. "Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance"
  - Z. Deng, W.S. Liu. "Scheduling real-time application in an open environment"
  - R. I. Davis, A. Burns "An Investigation into Server Parameter Selection for Hierarchical Fixed Priority Pre-emptive Systems"
  - G. Lipari, S. Baruah. "A hierarchical extension to the constant bandwidth server framework"

Mixed Criticality Systems (MCS)

- Real-time systems that integrate components with different levels of criticality on top of the same hardware platform
- The ***criticality*** is the designation of the level of ***assurance*** of the component with respect to malfunction (i.e., ***safety integrity levels - SIL*** in common functional safety standards such as IEC 61508)
- Introduced to satisfy stringent space, cost, weight and power consumption requirements, typical of real-time embedded systems
  - By avoiding the «multiplication» of dedicated computing units for the different functions and based on the different criticality levels, so allowing critical components to co-habit with non-critical ones
- The MCS model is currently recognized by main industry standards for real-time systems, such as AUTOSAR and ARINC

Mixed Criticality Systems (MCS)

- The main research challenge is to find a compromise between:
  - Partitioning or isolation: for safety guarantees
  - Sharing: for an efficient use of resources

- For these reasons, many contributions have been proposed lately, both
  - theoretical, e.g., on the modelling and verification of MCS
    - Vestal model
    - Run-time slicing
    - Dual-criticality
    - Hierarchical scheduling and related feasibility check
  - and practical, e.g., on hardware/software platforms for MCS
    - Mainly Separation Kernels and … **hypervisors**!!

Vestal's model

- Also known as *multicriticality* task model
- Based on the assumption that tasks' WCETs are dependent on the criticality level of the task
- A task will have a pessimistic WCET if it is critical
  - It will be subject to more stringent and thorough model-based analysis, which aim is to determine a strict upper bound to the execution time based on pessimistic assumptions
- The same code will have a shorter WCET if it is not critical
  - A runtime measurement of the execution time through monitoring is sufficient… but worst case conditions might not appear during measurement!

Vestal's model

- Every task τ_i is characterized by the following parameters:

$$(C_i, T_i, D_i, L_i)$$

modello sporadico, Ti è il interarrival minimo

- Where:
  - $C_i : N^+ \rightarrow R^+$ is an array of task's WCETs for different *criticality levels* l, such that: $C_i(l) \leq C_i(l+1)$
  - D_i is the relative deadline of the task
  - T_i is the period of the task (or minimum inter-arrival time: tasks can be sporadic)
  - L_i is the criticality level of the task

Vestal model: functioning modes

- The idea is to guarantee high criticality tasks in case of mulfunction
- The system executes trough variable *functioning modes* to give more guarantees to critical tasks, for instance:
  - The system starts at the minimum criticality level (mode: l) and schedules all tasks according to their $C_i(l)$;
  - As soon as a task with $L_i > l$ exceeds its $C_i(l)$ (e.g., due to malfunctioning, the system passes to the next criticality level (mode: l+1) where only the tasks with $L_i \geq l$+1 are guaranteed according to their $C_i(l$+1), whereas tasks with $L_i = l$ are no longer guaranteed, or discarded
- A simplified version is the ***dual criticality*** model:
  - Restricred to two criticality levels: HI and LO
  - It entails the evaluation of $C(HI)$ end $C(LO)$ for each task and the reject of LO tasks if the functioning mode is currently HI

Vestal: feasibility

- To let functioning modes work properly (i.e., to guarantee tasks at their criticality level for a given functioning mode), the feasibility of each task τ_i must be evaluated considering all tasks running at its criticality level L_i
- In other terms, , for each task τ_i and for a given scheduling algorithm A, it must be demonstrated that algorithm A is able to guarantee all the deadlines of task τ_i at its criticality level L_i

Feasibility: Multi-criticality RTA

- In the case of FTP scheduling (such as, Deadline Monotonic) a *multi-criticality* variant of the Response Time Analysis (RTA) can be used on single processors:

$$R_i = \sum_{j:\rho_j \leq \rho_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L_i)$$

Se minore uguale delle mia Deadline è feasible

- where ρ_i is the priority of the i-th task

Esempio
Abbiamo 3 task con i seguenti parametri:
Abbiamo due livelli di criticità, a seconda del livello i task hanno dei worst case execution time diversi
D1=T1=5, L1=A, C1A=2, C1B=2
D2=T2=4, L2=B, C2A=3, C2B=1
D3=T3=10, L3=B, C3A=3, C3B=2

Scheduliamo prima in ordine 1, 2 e 3 e poi mostriamo (nella slide successiva) che schedulando prima 2 non è fattibile

Considerando come priorità maggiore Task2 che effettivamente ha una deadline minore
R2= C2B =1 <= 4 OK
x= [5/5]*2
R1_0= C2A+C1A=5 R1_1= x + [R1_0/T1]*C2A=2+[5/4]*3=8 (considero tutti i task a livello di criticità dello stesso livello di criticità del task preso in analisi) 8>5 NON feasible

Proviamo a vedere cosa succede se considero Task1 a priorità maggiore

R1= 2 < =5 OK
R2_0= C1B + C2B = 2 + 1= 3 R2_1=1 + [3/5]2= 3 <=4 OK
R3_0 = C1B + C2B + C3B = 2+1+2 =5 R3_1= C3B + [R3_0/T1]C1B+[R3_0/T2]C2B= 2 + [5/5]2 + [5/4]1 = 6 --> fare l'ultima iterazione R3_2 e verificare che effettivamente è feasible

Vestal Method: cosa faciamo? Lezione del 11.11.2021 minuto 13

Non-optimality of DM

Abbiamo concluso che

- Deadline Monotonic is not optimal in the case of multi-criticality tasks

	τ_1	τ_2
T_i	2	4
D_i	2	4
L_i	B	A
C_{iA}	2	1
C_{iB}	1	1

- In the example, if A and B are two criticality levels
  - If the maximum priority is given to task 1, according DM, the set is not feasible
  - If the maximum priority is given to task 2, the set is feasible

How to schedule multi-criticality tasks?

- Possible solutions:
  - DM scheduling with ***run-time slicing***
  - **Optimal priority assignment** with the *Audsley method*

Run-time slicing

- The idea is to let high-criticality task to have the smallest period, in order to be assigned the highest priorities according to DM…
- … by slicing long critical tasks into subtasks

- Basically, assuming $D_i = T_i$, a high-criticality task can be split into n fragments with $T'_i = T_i/n$ and $C'_i = C_i/n$ with n integer big enough to reduce the period of the task to a value smaller or equal to the period of any other task with lower criticality.
  - all the tasks with high criticality will have higher priority than low priority tasks
  - This is also called *criticality monotonic*

- This can be extended to the case $D_i < T_i$

- But it introduces the complexity of the *slicing* and a run-time support to suspend the task when it reaches C'_i

Priority assignment: the Audsley's method

- Optimal priority assignment of the priority, based on the following lemmas:
  - **Lemma 1**: the worst case execution time of a task τ_i can be determined (for instance with RTA) solely by knowing the tasks with higher priority with respect to τ_i but without knowing the specific rule to assign task priority
  - **Lemma 2**: if a task is schedulable considering a given priority assignment it will remain schedulable, even if it will receive a higher priority
- Can be used with multi-criticality RTA, but can be extended to other situations for fixed priority assignment

The Audsley's method

```
Audsley's_optimal_priority_assignment (Γ, S) { // Γ task set
                                               // S: schedulability test
  for (each priority level k in n:1) { // from the lowest level
        for (each unassigned task τ in Γ) {
              if (τ is schedulable with S, with all unassigned tasks
                  assumed to have priorities > k) then {
                    assign τ to priority k;
                    break;
              }
        }
        return UNSCHEDULABLE;
  }
  return SCHEDULABLE;
}
```

In pratica scansiono tutti i task con una ricerca lineare e se il task in esame è schedulabile applico il lemma 2 e gli assegno proprio la priorità k

Vestal's model: limitations

- The criticality concept has a different meaning in Safety Critical standards:
  - A given SIL can be assigned to a given functionality or component that can involve different tasks
  - If a component has high failure severity but low failure probability, it could be assigned a low criticality level
    - And thus, in a dual criticality system, it will be LO criticality and aborted if it the system is not able to schedule HI tasks…. with severe consequences!
    - In other terms, even low criticality tasks must execute, they cannot be simply aborted
- Certification standards require *separation* and *no interference* between functions at a different criticality level
  - If this is not the case, all the functions must be developed with the highest SIL → higher costs!
  - Achieving a compromise between separation and resource sharing is still an open problem

Hierarchical scheduling

- A different approach to achieve mixed-criticality with better separation
- Two levels of scheduling
  - LOCAL: Scheduling of tasks belonging to an application A_k
  - GLOBAL: Scheduling of applications, performed by the operating system

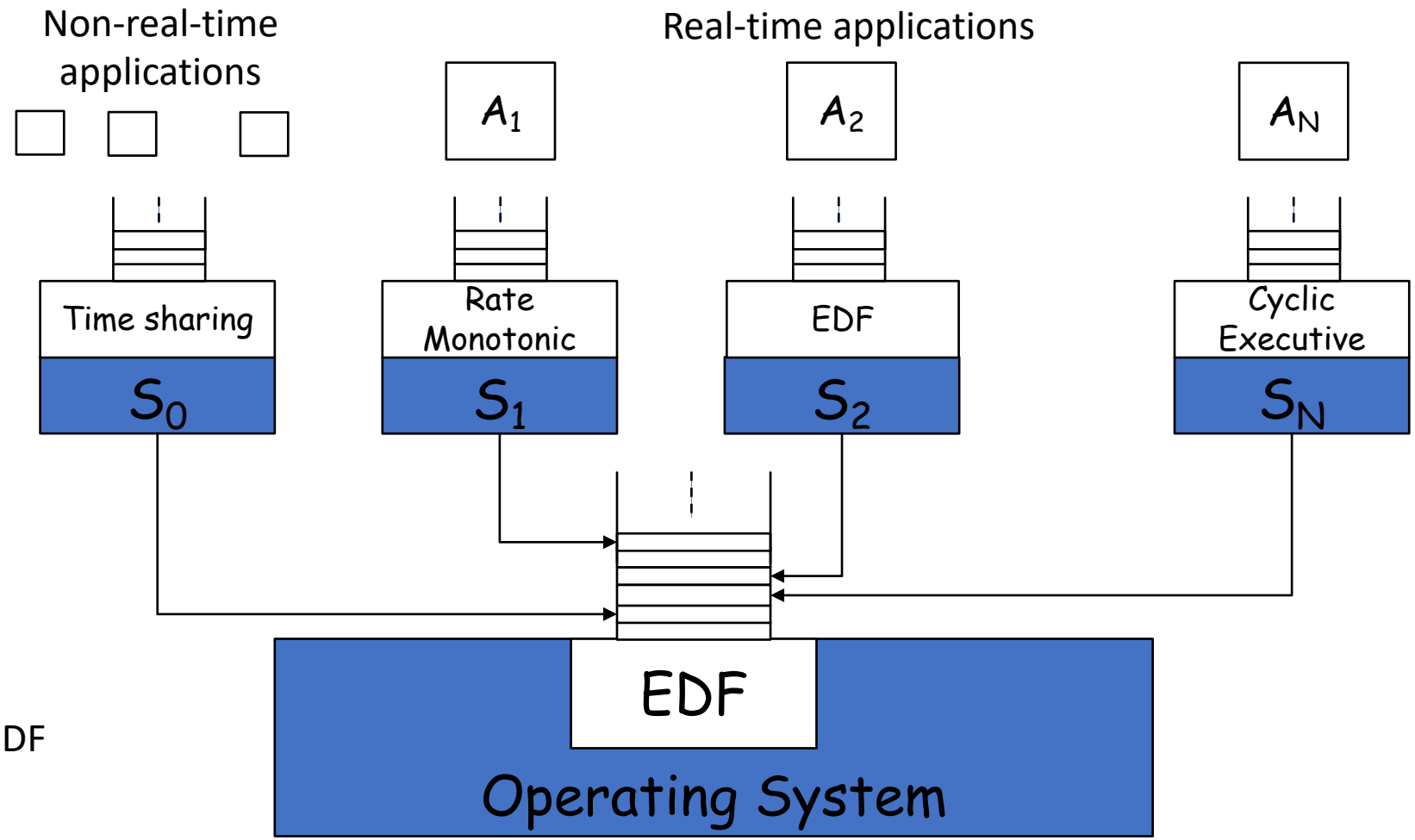
- The applications are served each one by an aperiodic server (Polling Server, Deferrable Server, Constant Bandwidth Server, …)
  - Applications with different criticality levels will receive each one its server and its share of guaranteed CPU bandwidth

- The operating system manages the servers and schedules them
  - Recharges the server budget, controls the available capacity, …

Hierarchical scheduling: a selection of approaches

- Open system architecture
  - (Z. Deng, W.S. Liu. “Scheduling real-time application in an open environment)
- Hierarchical FTP Preemptive Systems
  - R. I. Davis, A. Burns “An Investigation into Server Parameter Selection for Hierarchical Fixed Priority Pre-emptive Systems”
- Hierarchical FJP Preemptive Systems (Hierarchical CBS)
  - (G. Lipari, S. Baruah. “A hierarchical extension to the constant bandwidth server framework”)

Open system architecture



Local scheduling: Variable

Global scheduling: EDF

Dimensioning the server capacity

- Consider a server S_k, with a scheduling quantum q, that has to serve the jobs of the application A_k, with the following parameters:
  - σ_k : A_k jobs' total utilization
  - δ_k : shortest relative deadline of the jobs in A_k
- Jobs of Ak are feasible if the server is a TBS or CBS with bandwidth

$$U_k = \delta_k \sigma_k / (\delta_k - q)$$

the quantum has to be lower than the shortest deadline (how lower depends on σ_k and introduced overheads). Ideally, if it is zero, the server utilization equals the application utilization

Introducing Resources

- The Open System Architecure allows also to model shared resources.
- At local level, using classical solutions (Priority Inheritance, Priority Ceiling,…) managed by the application-level scheduler
- At global level, using the Non-Preemtive Protocol
  - When a job of an application A_k requires a global resource, the application and the respective server becomes non-preemptable
  - The OS monitors the application to assure that the critical section will not overcome a predermined value B_k

Server Schedulability, with resources

- A hierarchical scheduling system, using a TBS or CBS as a server, is schedulable with EDF if the total bandwidth U_t of all servers satisfies the condition:

$$U_t \leq 1 - \max_i \{B_i/\delta_i\}$$

- with B_i blocking time of the longest global (non-preemptive) critical section executed by any other server different from S_i
- and δ_i the shortest relative deadline among all the tasks executed by S_i

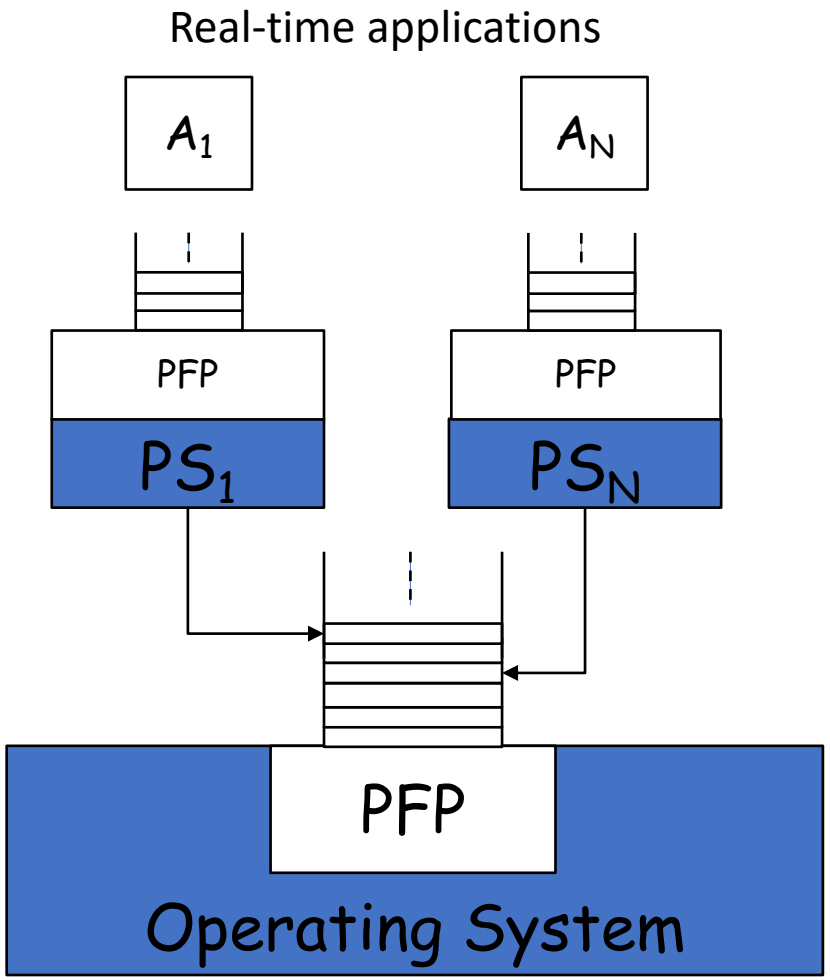
Admission test

- An application A_k that wants to be admitted, provides:
  - The scheduling algorithm and the e la type of task (preemptive, non preemptive, periodic, sporadic, ...)
  - The total utilization of A_k jobs utilization $\sigma_k < 1$
  - The maximum lenght L_k of all the global (non preemptable) critical sections
  - The shortest relative deadline δ_k among all its jobs
- In case of sporadic nonpredictable tasks, U_k is determined as $\delta_k \sigma_k / (\delta_k - q)$
- A_k is accepted if: $U_t + U_k + \max_{1 \le j \le N} \{B_j / \delta_j\} \le 1$, where N is the total number of applications including Ak and and $B_j = \max_{i \ne j} \{L_i\}$
- If A_k is admitted, then $U_t = U_t + U_k$ and a server S_k with the specified bandwidth is created

Hierarchical FTP preemptive systems

- Both the local and the global schedulers are preemptive fixed-priority (PFP)
- The task model is the constrained-deadline sporadic $\tau_i = (C_i, D_i, T_i)$
- Each server is a Polling Server (PS) with a capacity C_S and a period T_S
- A task is feasible if its response time R_i is lower than D_i

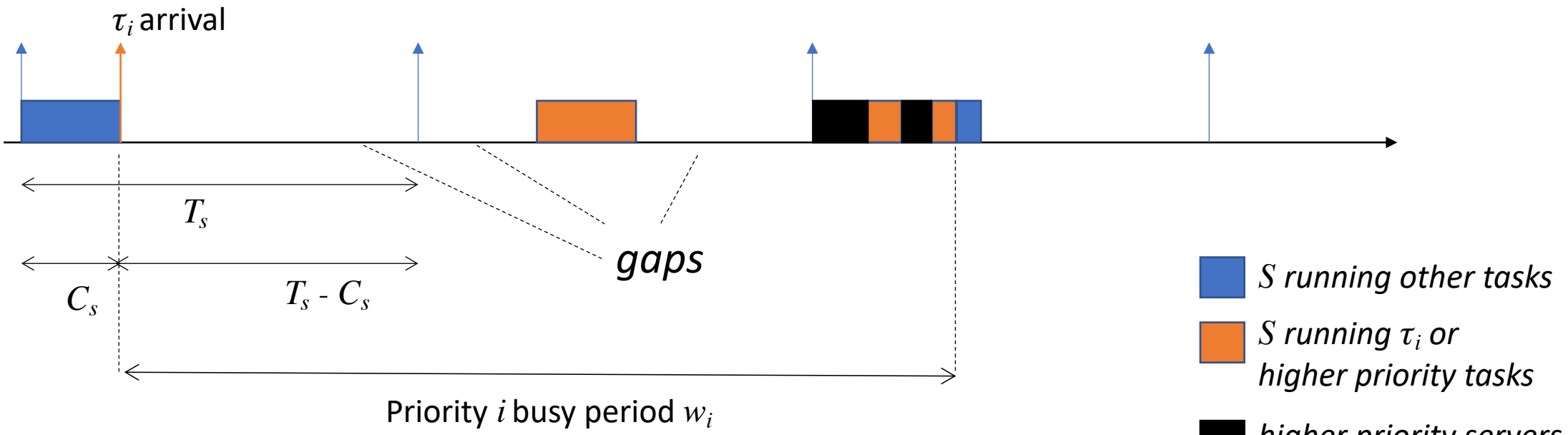
We have to find a way to evaluate R_i taking into account both local and global interferences



How to determine the response time in hierarchical FTP systems?

- We can adopt the **Response Time Analysis**, assuming the following worst conditions for a task τ_i running on a Polling Server S:
  - The capacity of S is exhausted by lower priority tasks as early in its period as possible
  - Task τ_i and all higher priority tasks in the application allocated on S arrive just after the server's capacity is exhausted
  - The server's capacity is replenished at the start of each subsequent period, however further execution of the server is delayed for as long as possible due to interference from higher priority servers

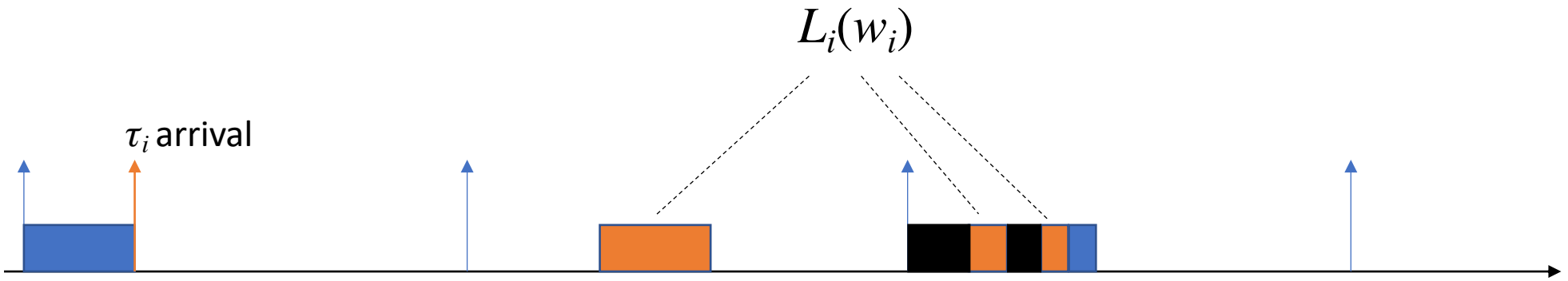
Sources of interferences



Busy period

- a priority level *i* **busy period** w_i is defined as an interval of time during which there is outstanding task execution at priority level *i* or higher
- Some notations:
  - $w_i(L)$: the time (the window w) that the application's server can take to execute a given load L
  - $L_i(w)$: the total task executions, at priority level *i* and above, released by the application within a time window of length *w*

Sources of interferences: execution



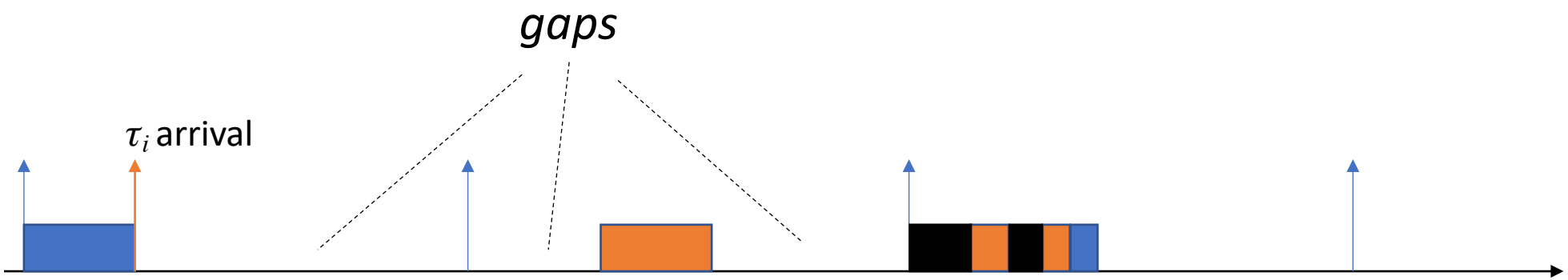
$L_i(w_i)$ can be evaluated with the classical response time analysis: Ji jitter of attivation

$$L_i(w_i) = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j$$

Where:

- $hp(i)$ is the set of tasks with higher priority than τ_i
- J_j is the release jitter of the j-th task

Sources of interferences: gaps

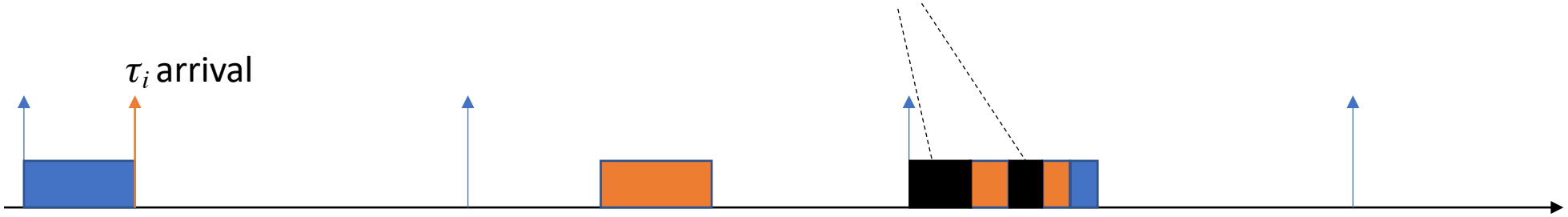


Not including the final server period, the lenght of server gaps is given by:

$$\left(\left\lceil \frac{L_i(w_i)}{C_S} \right\rceil - 1\right)(T_S - C_S)$$

Sources of interferences: other servers

Interference from higher priority servers



The extent to which the busy period w_i extends into the final server period is given by:

$$w_i - \left(\left\lceil \frac{L_i(w_i)}{C_S} \right\rceil - 1 \right) T_S$$

On this extent, we can evaluate the interfernce from any higher priority server X with capacity C_X and period T_X

… wrapping up

- The full extent of the busy period can be evaluated recursively:

Li(wi) gaps balck boxs

$$w_i^{n+1} = L_i(w_i^n) + \left(\left\lceil \frac{L_i(w_i^n)}{C_S} \right\rceil - 1\right)(T_S - C_S) + \sum_{\substack{\forall X \in hp(S) \\ servers}} \left\lceil \frac{\max\left(0, w_i^n - \left(\left\lceil \frac{L_i(w_i^n)}{C_S} \right\rceil - 1\right)T_S\right)}{T_X} \right\rceil C_X$$

è una formula iterativa: wi sia a sinistra che a destra

- Where $hp(S)$ is the set of servers with higher priority than S.
- The recursion can be initialized with $w_i^0 = C_i + \left(\lceil C_i / C_S \rceil - 1\right)(T_S - C_S)$
- and ends either when $w_i^{n+1} = w_i^n$ or when $w_i^{n+1} > D_i - J_i$, in which case the task is not schedulable

How to select the parameters of the servers?

- We have to set 3 parameters for each server: priority, capacity, period
- We can fix two of them and optimally find the other
- Or we can use greedy search algorithms

Modified Audsley's priority assignment

- If budgets are known for each server, in terms of capacity/period fractions, priority can be assigned with a modified version of the Audsley algorithm, where the schedulability is evaluated trough the recursive formula of busy periods

```
OPTIMAL SERVER PRIORITY ASSIGNMENT ALGORITHM
for each priority level, lowest first
{
    for each unallocated server
    {
        if the server and its tasks are
        schedulable at this priority level
        {
            allocate server to this priority
            break (continue with outer loop)
        }
    }
    return unschedulable
}
return schedulable
```

Optimal capacity allocation

- Similarly, if priority and periods are known, we can seek for optimal server capacities

```
OPTIMAL SERVER CAPACITY ALLOCATION ALGORITHM
for each server, highest priority first
{
    binary search between 0 and the
    server period for the minimum capacity
    Z that results in the server and its
    tasks being schedulable.
    if no schedulable capacity found
    {
        exit system not schedulable
    }
    else
    {
        set the capacity of the server to Z
    }
}
```

Hierarchical-CBS

- Hierarchical extension of the CBS: H-CBS
- Tasks are partitioned in subsystems $S_1, S_2, \ldots S_N$ sharing the same processor
- Each subset contains the tasks of an application, to be guaranteed in terms of performance and isolation (for incremental development & certification)

H-CBS: example

- Suppose to have three task, J_1, J_2 e J_3, each with bandwidth 1/3, with J_1 e J_2 belonging to the set (application) S_1, and J_3 to the set S_2.
- If all 3 tasks execute continuosly on a CBS, S_1 receives 2/3 of the processor, and S_2 1/3, as expected.
- If J_2 becomes idle, the CBS will end to assign 1/2 of the processor to J_1 and 1/2 to J_3, hence violating the constraint to assign an overall 2/3 of bandwidht to application S_1
- H-CBS extends CBS to guarantee the bandwidth of a task set seen as an overall, application, independently from the number of tasks currently executing in a given instant of time

H-CBS: system model

- System composed by a task set τ of sporadic tasks, each with its own bandwidth, U_j, its WCET C_j and a minimum inter-arrival time T_j
- The set τ is partitioned in the subsets $S_1, S_2, \dots S_N$, such that: $S_1 \cup S_2 \cup \dots \cup S_N = \tau$ and $S_i \cap S_k = \{\}, \forall i \neq k$
- For each subset S_i it is $U(S_i) = \sum_{J_j \in S_i} U_j$ and, globally, $\sum_{i=1}^{N} U(S_i) \leq 1$
- The aim of H-CBS is to guarantee both the execution of the individual tasks, as if they were executing each on a CPU with bandwidth U_j, and the execution of subsets, as if they were executing on a CPU with bandwidth $U(S_i)$

H-CBS: functioning

- For each task, J_j, H-CBS manages two variables:
  - D_j, task deadline
  - V_j, virtual time of the task
- Tasks are normally scheduled with EDF
- V_j is a measure of how much bandwidth reserved to J_j has been consumed in a given instant
- To guarantee $U(S_i)$, H-CBS introduces a *reclaim* algorithm for the bandwidth unused by tasks $J_j \in S_i$ due to:
  I. Tasks of S_i not executing in a given interval
  II. Tasks of S_i that terminate before using all the reserved bandwidth
- The unused bandwidth is assigned to a task $\in S_i$ (a task in execution and with the highest priority in S_i)

H-CBS: variables update

- When a new job of J_j arrives at time *t*:
  - $V_j = t$ and $D_j = V_j + T_j$
- When the current job terminates and there is already a new ready job of J_j :
  - $D_j = V_j + T_j$
- When $V_j = D_j$, the deadline is delayed of a period (to guarantee the task bandwidth U_j):
  - $D_j = D_j + T_j$

H-CBS: variables update

- if $J_j \in S_i$ is the task with the current highest priority in S_i:

$$\frac{dV_j}{dt} = \begin{cases} \frac{1-\rho_i}{U_j}, & if\ J_j\ is\ executing \\ -\frac{\rho_i}{U_j}, & if\ J_j\ is\ not\ executing \end{cases}$$

  - where ρ_i is the unused bandwidth in S_i, equals to the sum of the bandwidths of currently inactive tasks in the set
    - initially $\rho_i = U(S_i)$
    - When $J_j \in S_i$ starts to run, $\rho_i = \rho_i - U_j$
    - When $J_j \in S_i$ ends, $\rho_i = \rho_i + U_j$

- If $J_j \in S_i$ ends at a time t before using all its budget, it transfers its residual budget to the task J_l with the current highest priority in S_i (if $V_j < t$):

$$V_l = V_l - \frac{(t - V_j)U_j}{U_l}$$