논문세미나

A methodology for static and dynamic scheduling of automation tasks in reconfigurable production systems

- 일시 : 2013. 02. 18

신 정 훈

Contents

선정사유

- 자율적응 과제를 하면서
  - 제어를 공부하는 사람들이 보는 컨트롤 측면에서의 스케줄링에 대한 궁금증
  - 스케줄링 체계에 대한 시스템 아키텍쳐 서술에 대한 논문 서치(for ISGMA)

Introduction

DCS(distributed control systems)

- DCS(distributed control systems)
  - a number of interrelated control modules encapsulating portion of control governing the behavior of system mechatronic devices
  - instrumental when production systems undergo physical and logic adaptations and reconfigurations

Traditional vs Real-time System

- Traditional system
  - control engineer assumes that the control task sequencing is optimized by **the solver of the control software tool** utilized for the implementation
    ➔ most of the software tools are equipped with **very simple internal policies** for sequencing tasks
  - production scheduling problem - traditionally addressed by production engineering
- Real-time system
  - control task scheduling - traditionally addressed in the **computer science field** under the name of "scheduling of real time systems"
  - deadlines or other explicit timing **constraints are attached to task**
  - the correctness in execution and performance are tightly interrelated
- Each production task to be realized by the robot is decomposed in a number of control tasks and signals to be processed and executed on the robot real-time system.

major innovation of the proposed approach

- the integration of static and dynamic control scheduling algorithms
- the modeling of different types of control tasks and the incorporation of technological constraints between tasks
- the application of the approach to a DCS(distributed control system) operating in an existing production solution, thus modeling a number of control tasks reasonably close to reality.

Real-time system scheduling basics and literature review

Task

- Task
  - have a **duration** normally named worst case execution time(**WCET**; indicated with **C**)
  - have a **response time** (indicated with **R**) representing the exact instant in which the task is actually concluded
  - task **deadline** - The **time constraint** for tasks
- periodic, aperiodic
  - Task can be **periodic** when the task **starting time is replicated for every period**
  - **aperiodic** - the starting and ending time of the task can flexibly vary.(activated irregularly)
- A group of tasks can be **synchronous** when the **first instances of these tasks have all the same starting time** ↔ **asynchronous**
- deadline
  - **hard deadlines** for real-time tasks imply a strict fulfilling of the temporal constraint whose disrespecting leads to a **violation of vital safety of the system**
  - **soft deadlines** represent a more flexible constraint, thus involving **nothing vital** but **with a penalty**
- **capacity** - determines the amount of processor capacity utilized to execute the tasks
- **preemption**
  - Tasks can be preemptable if their execution can be **interrupted for a certain time**.
  - This interruption is generally **not preferable** and, for this reason, a task interruption is frequently associated to a **penalty**

Static and Dynamic algorithm

Static algorithm

- produce a task sequence which is then **assumed to be successfully executed by the resources**
- two very basic algorithms
  - **EDF**(Earliest Deadline First) – 최단마감우선처리 : deadline 이 가장 이른 것 부터 처리(earliest due date)
  - **RMA**(Rate Monothonic Analysis) – 비율단조분석 : 수행주기가 최소인 프로세스에 최고 우선순위

Dynamic algorithm

- adapt the scheduling plan on the **basis of the actual execution**, thus closing the loop with the actual execution performed by shop-floor resources
- roughly classified under two major categories
  - **completely reactive** scheduling strategies - **only after detecting a misalignment** between nominal and actual task execution
  - **predictive-reactive** scheduling strategies - **use the information gathered from the actual execution** analysis to build some sort of patterns for making more robust the scheduling process to future potential changes

The ISA approach

- The static scheduler
- The dynamic scheduler
- The diagnoser

ISA(Integrated Scheduling Architecture)

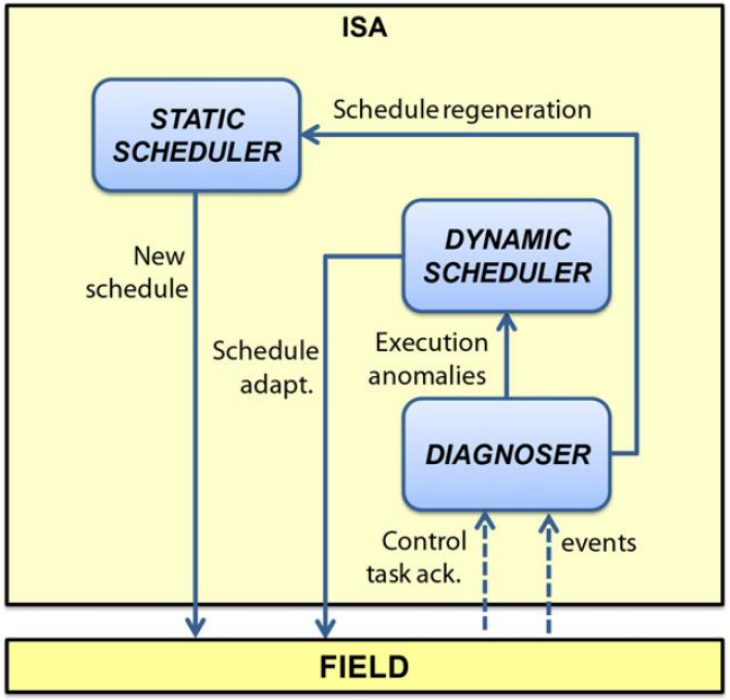

Fig. 2. ISA approach.

- three main modules
  - **the static scheduler, the dynamic scheduler and the diagnoser**
  - the static scheduler **generates a nominal scheduling plan** that sequences the tasks while minimizing the idle times
  - While control tasks are running on the physical system, the **diagnoser collects information** about the task **actual execution times** together with any particular **unforeseen event**
  - The dynamic module is responsible for the **on-line schedule adaptations** based on a **misalignment between nominal and actual** control tasks execution

- three major novelties of ISA
  - it **integrates additional task scheduling** functionalities to the **control software**
  - it consists of a unique **software infrastructure comprehending three modules** each one realizing several control functions both static and dynamic
  - it is designed and developed in order to **enable the general applicability** as it can be deployed on different control solutions with no constraints or requirements of specific control software platforms and performance

The static scheduler

- Notations(1)
  - r : manufacturing resources
  - j : jobs defined by [startj; endj]
  - $Sc = \{sc_1, \ldots, sc_J\}_r$: manufacturing schedule
  - pr : priority value
  - τ_i : ith automation task(τ_{ki} , k∈N+)
  - s_i : Task starting time
  - C_i : Worst case execution time (WCET) representing the average nominal duration of the tasks;
  - T_i : period;
  - Cap_i : resource capacity absorbed to execute task i.
  - start time of τ_{ki} is given by $s_{ki} = s_{0i} + (k-1)T_i$ where s_{0i} represents the starting time of the first instance of τ_i;
  - the parameter Exe_i determines the nominal rate of the period T utilized for the execution of task i; it is given by $\sum_i(C_i/T_i)$.
  - The parameter Cap_r defines the total resource capacity.
  - α : Task preemption penalty
  - $Np(\tau_{ki})$: the exact number of preemptions of τ_i in τ_{ki}
  - $C_{ki}=C_i+Np(\tau_{ki})* \alpha$: WCET after a preemption
  - R_{ki} : the response time of τ_{ki}
  - h_i = LCM{(LCM)τ_w ;(T_i)} : hyperperiod. LCM(Least Common Multiple;최소공배수)
  - σ_i : The number of times a specific task is started within h_i
  - $State_{ki}$: captures the time unit already filled(by the instance kth of task τ_{ki}) in the period T_i. it is possible to identify the available time slots which can be further occupied by other tasks.
  - $\{State^1_i \ldots State^{\sigma_i}_i\}^{BeforeSched}$: the state space before a scheduling action.
  - $\{State^1_i \ldots State^{\sigma_i}_i\}^{AfterSched}$: the state space after a scheduling action.
  - A_{ki} : Available : The subset of available time slots in the (T_i-$State_i$)
  - $State^{intermediate}(\tau_i)$ synthesizes in a single vector the availabilities resulting from the allocation of tasks τ_w, with w = 1, ..., i-1.
  - $$Exe_i^* = \sum_{i=1}^{l} \frac{1}{\sigma_i}\left(\sum_{k=1}^{\sigma_i} \frac{C_i^k}{T_i}\right) = Exe_i + \sum_{i=1}^{l} \frac{1}{\sigma_i}\left(\sum_{k=1}^{\sigma_i} \frac{Np(\tau_i^k)\alpha}{T_i}\right)$$

    the exact value of period T utilization for the execution of task τ_i. Given l the level of the scheduling

The static scheduler

- Notations(2)
  - the binary matrix OP : the technological precedence constraints: the single entity of this matrix pre_{iw}
    pre_{iw} =1 task τ_i and τ_w can be concurrently execute
    pre_{iw} =0 task τ_i and τ_w must be sequentially processed
    (i, w = 1, . . ., i-1 with i≠w).
  - the binary matrix $OP^{resource}$: lists the association between tasks and the resources where they are executed. The single entity of this matrix p_res
    p_res=1 if the generic task is associated to the generic resource and 0 otherwise.
  - $idle_i$ =(1- Exe_i) : idle time

The static scheduler

- 5 Main steps
  - **construction of the schedulable set of tasks** which involves ordering tasks coherently with a specific **policy** and computing all the fundamental **parameters**
  - **verifies the schedulability conditions** on the basis of the sets built in the previous phase and three major feasibility constraints(related to the processor availability, the processor capacity and the actual execution of previous tasks)
  - **generates a valid schedule** by sequencing the tasks while minimizing the idle times
  - **computes the exact values of parameters** based on the exact number of preemptions and exact allocations
  - **verifies the coherency between the production scheduling and the automation one**, i.e. it checks whether the exact response times affect the deadlines of production jobs scheduling.

The static scheduler–Step1

Step 1. Construction of schedulable set

(1.1) For each job j assigned to the generic resource $r = \{1, \ldots, R\}$ in the scheduling $Sc = \{sc_{01}, \ldots, sc_{0J}\}_r$ select $\{\tau_1, \ldots, \tau_n\}_{j,r}$.

(1.2) For each $i = \{1, \ldots, n\}$ list τ_i with RMA policy with respect to technological precedence based upon OP matrix.

(1.3) Given the generic τ_i and τ_u where $\tau_i \in r$ and $\tau_u \in rr$ for $r,rr \in \{1, \ldots, R\}$ with $r \neq rr$, tasks τ_i and τ_u can be run in parallel iff $p\_res(\tau_i, r) = p\_res(\tau_u, rr) = 1$.

(1.4) For each i and for each r compute $h_i = LCM\{(LCM)\tau_w; (T_i)\}$.

(1.5) For each i and for each r compute $\sigma_i = H_i/T_i$.

(1.6) For each i and for each r compute $Exe_i = \Sigma_i C_i/T_i$.

(1.7) Compute $Exe^*_{(i-1)}$ which is the exact utilization rate for task $(i-1)$.

(1.8) For each i and for each r generate $\{State^1_{intermediatei}, \ldots, State^{sp(\tau_i)}_{intermediatei}\}_r$.

(1.9) For each i and for each r starting from $\{State^{sp(\tau_i)}_{intermediatei}\}_r$, generate $\{State^1_i \ldots State^\sigma_i\}^{BeforeSched}$ and the related set $\{A^1_i, \ldots, A^\sigma_i\}$ and $C^1_i, \ldots, C^\sigma_i$.

job을 resource에 할당(job의 task를 resource에 할당)

OP matrix(동시가공가능여부) 를 참조하면서 기술적 선행관계(process plan등)를 고려하여 RMA policy 적용

동시가공가능조건 확인

hyperperiod계산(최소공배수 계산)

h_i에서 task의 시작횟수 계산

task i가 T_i에서 활용된 비율 계산

$Exe^*_{(i-1)}$ 계산

중간 availability 확인

T_i의 availability 확인

The static scheduler–Step2

Step 2. Schedulability condition

(2.1) τ_i is schedulable if:

$$\begin{cases} C_i \in A_{k(i)} \\ Exe^*_{i-1} + \dfrac{C_i}{T_i} \le 1 \\ \sum_{j=1}^{i-1} Cap_j + Cap_i \le Cap_r; \quad j,\ i = 1, .., n \end{cases}$$

schedule 가능 상태 확인

processor availability

actual execution of previous tasks

processor capacity

(2.2) If τ_i is schedulable then go to Step 4.

(2.3) Else set $(h)_{\tau i} = (h)_{\tau w}$, $s_i = 0$ and $Exe_i = 0$.

The static scheduler–Step3

Step 3. Definition of valid schedule

(3.1) For each $i = \{1, \ldots, n\}$ compute $Np(\tau_{ki})$.

(3.2) For each i, compute $Exe_i = \sum_i C_i/T_i$.

(3.3) For each i, compute idle times $idle_i = (1 - Exe_i)$.

(3.4) For each $i = \{1, \ldots, n\}$ compute R_{ik}.

(3.5) For each $i = \{1, \ldots, n\}$ compute $Responsetime_i = max(R_{ik})$.

(3.6) For each $i = \{1, \ldots, n\}$ and each $k = \{1, \ldots, \sigma\}$, if $max_{ik}(R_{ik}) \leq T_i$ and $C_{k(i)} \leq A_{k(i)}$ then:

(3.6.1) generate $\{State_i^1 \ldots State_i^{\sigma}\}^{BeforeSched}$

(3.6.2) for each i, compute Exe_i^*.

(3.6.3) for each i, compute idle times $idle_i^* = (1 - Exe_i^*)$.

(3.6.4) go to Step 1 and increment to $(i + 1)$

(3.6.5) when $i = n$ go to Step 4.

(3.7) For each $i = \{1, \ldots, n\}$ and each $k = \{1, \ldots, \sigma\}$, if $max_{ik}(R_{ik}) \geq T_i$ and/or $C_{k(i)} \geq A_{k(i)}$ then set $(h)\tau_i = (h)\tau_{(i-1)}$, $s_i = 0$ and $Exe_i = 0$.

선점 task 계산

활용률 계산

idletime 계산

task ik의 Response time 계산

i번째 max Response time 계산

a

state space 생성

Exe^*_i 계산

$idletime^*$ 계산

step 1으로 돌아가서 i를 1증가

i에 대해 모두 수행하였으면 step4로 감

최대 응답시간이 period를 초과, avg nominal duration of task가 available time slot보다 크면(할당 불가) 파라메터 수정

The static scheduler–Step4

Step 4. Definition of exact schedule

(4.1) For each $i = \{1, \ldots, n\}$ and each resource r, compute $(Exe_i)r^* = \sum_{i=1}^{n} (Exe_i)_r$

(4.2) For each $i = \{1, \ldots, n\}$ and each resource r, compute $(idle_i)r^* = (1 - (Exe_i)_r^*)$

(4.3) Compute the exact schedule $\mathbf{S} = \{s_{01}, \ldots, s_{0n}\}$

(4.4) return "set of start times of the first instance of all tasks is $\mathbf{S} = \{s_{01}, \ldots, s_{0n}\}$".

(4.5) go to Step 5.

resource 별 활용률 계산

리소스별 idle time 계산

확정된 수치에 기반한 스케줄 생성

생성된 스케줄을 기준으로 task의 첫 번째 인스턴스의 시작시간 반환

The static scheduler–Step5

Step 5. Evaluate the coherency between mfg and automation schedules

(5.1) For each τ_i, compute $(s_{0i} + R_i)_{r,j}$

(5.2) For each job j and for each resource r if $\{max_i[(s_{0i} + R_i)^j] < end_j\}_r$ then return "the automation schedule suites the production schedule" else

(5.3) compute $delay = \{max_i[(s_{0i} + R_i)_j] - end_j\}_r$

(5.4) display(['$delay$ = '$delay$]).

End.

task 의 첫 번째 인스턴스의 시작시간 + 응답시간 계산

각 job별로 5.1의 계산값중 최대값이 종료시간을 넘지 않으면 맞는 스케줄임

job별로 지연시간 계산

지연시간 보여줌

The dynamic scheduler–Step6

- dynamic scheduling module performs three major adjustments
  - **Adapt** the worst case execution time($\mathbf{C_i}$) with the **actual values**
  - **Updates the priorities** of automation tasks
  - Generates **minor sequence adjustments** on the basis unforeseen events

Step 6. Tuning of WCET

(6.1) While executing $\varsigma = \{s_{01}, \ldots, s_{0n}\}$, for each resource r and job j and for each automation task τ_i with $i = \{1, \ldots, n\}$, track the actual value of worst case execution time entitled C_i^*.

(6.2) if $C_i = C_j^*$ is true stop.

(6.3) if $C_i = C_i^*$ is false go to Step 1.

Static module에서 생성한 스케줄을 실행하는 중에 WCET의 실제값 추적

WCET의 실제값이 Static 모듈의 계산값과 같으면 stop

WCET의 실제값이 Static 모듈의 계산값과 다르면 step1으로 가서 static scheduling 재실행

The dynamic scheduler–Step7, 8

Step 7. Adapt the task priority value

(7.1) While executing $\varsigma = \{s_{01}, \ldots, s_{0n}\}$, for each resource r and job j and for each automation task τ_i with $i = \{1, \ldots, n\}$, it is set $pr(\tau_i)! = pr^*(\tau_i)$, then:

(7.2) for each $i = \{1, \ldots, n\}$ list τ_i with RMA policy with respect to technological precedence and priorities.

(7.3) go to Step 1.3.

Static 스케줄 실행중에 자동화작업의 우선순위에 이상이 없으면(?)

각각의 task에 RMA policy를 적용함(기술적 선행조건과 우선순위를 고려)

step1의 1.3으로 가서 static scheduling 재실행

Step 8. Dynamic adaptation based on execution gap Δ

(8.1) For each resource r and job j and for each automation task τ_i with $i = \{1, \ldots, n\}$ of the schedule $\varsigma = \{s_{01}, \ldots, s_{0n}\}$, compute $\Delta = \{C_{ki}^{actual} - C_{ki}^{nominal}\}$.

(8.2) If $\Delta \in \{gap^-:gap^+\}$ stop.

(8.3) If $\Delta > gap^+$ or $\Delta < gap^-$, list τ_i with LEF policy with respect to technological precedence and priorities.

(8.4) go to step 1.3.

WCET의 static 결과값과 실제 값의 gap을 계산

gap이 범주안에 있으면 종료

gap이 범주를 벗어나면 각각의 task에 RMA policy를 적용함(기술적 선행조건과 우선순위를 고려)

step1의 1.3으로 가서 static scheduling 재실행

The dynamic scheduler–Step9

Step 9. Execute a recovery for a failure

(9.1) While executing $\varsigma = \{s_{01}, \ldots, s_{0n}\}$, for each resource r and job j and for each automation task τ_i with $i = \{1, \ldots, n\}$, a failure occurs do:

(9.1.1) select the automation tasks which are involved by the failure and mark as τ_i^f with $i = \{1, \ldots, n\}$.

(9.1.2) for each τ_i^f, associate a worst case execution time entitled C_i^f equal to very high constant value.

(9.2) select list of new tasks related to extraordinary maintenance to be introduced in the sequence and mark τ_i^{xm}.

(9.3) for each τ_i^{xm} with $i = \{1 \ldots, n\}$, associate the related value of priority pr^{xm}.

(9.4) generate a new task set $\{\tau_1, \ldots \tau_n\}_{j,r}^{new}$ including existing tasks and new tasks.

(9.5) load new matrices OP and $OP^{parallel}$ from controller.

(9.6) for each $i = \{1, \ldots, n\}$ list the tasks of the set $\{\tau_1, \ldots \tau_n\}_{j,r}^{new}$ with RMA policy with respect to technological precedence and priorities.

(9.7) go to Step 1.3.

failure 가 발생하면

failure에 관계된 task를 골라서 표시함

failure에 관계된 task들에 대하여 WCET를 별도 표시하고 아주 큰 상수로 초기화 함

수행할 비정기 유지보수와 관련된 task를 뽑아 별도 표시

9.2의 task들에 대하여 우선순위를 부여

새로운 task set 생성

OP관련 matrix를 갱신

새로운 task set에 RMA policy 를 적용함(기술적 선행조건과 우선순위를 고려)

step1의 1.3으로 가서 static scheduling 재실행

The diagnoser

- The diagnoser includes
  - an **abstract model** of the shop-floor main variables(nominal value)
  - the **nominal behavior** of the control resources
  - the **decision processes and policies** for handling unforeseen events

Step 10. Real-time diagnosis

- Measuring of actual C_i^* and comparison with the nominal value.
- Measuring of actual R_{ki}^{actual} and comparison with the nominal value $R_{ki}^{nominal}$.
- Detecting warning and alerts from resource PLCs and sensors.

WCET(Worst Case Execution Time)의 계산값과 실제값 비교

Response time의 계산값과 실제값 비교

리소스의 PLC와 센서로부터 오는 경고와 알람 감지

Description of the industrial test case

- Case 설명
- Experimental testing of the ISA approach

case 설명

- case
  - The **considered production line** has been **designed and configured in 2004** for the manufacturing of customized **shoes**
  - system Goal : **automate a very complex manufacturing process** which is traditionally manually handled for most of the operations while **guaranteeing very high levels of shoes quality and production throughput**
  - A peculiar feature of the system concerns the transportation system, **entitled molecular line(6 turns)**

Fig. 3. Customized shoe factory in ITIA-CNR Vigevano.

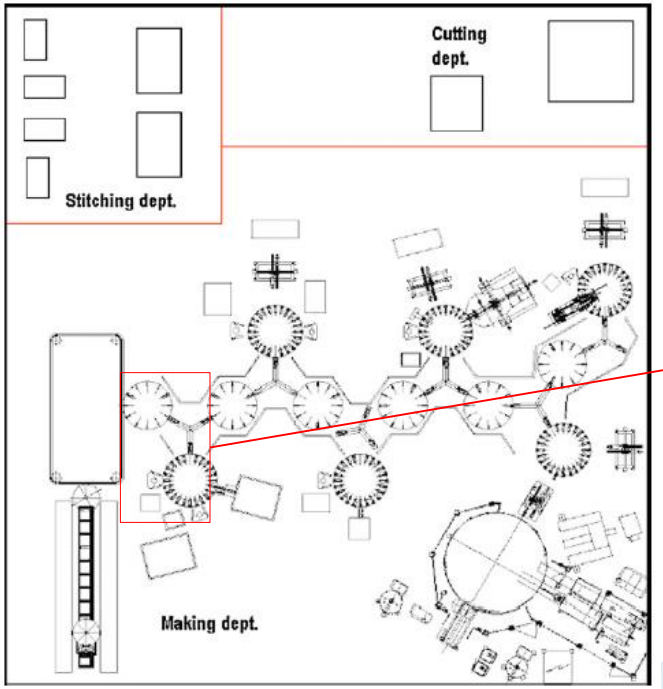

Fig. 4. Molecular line.

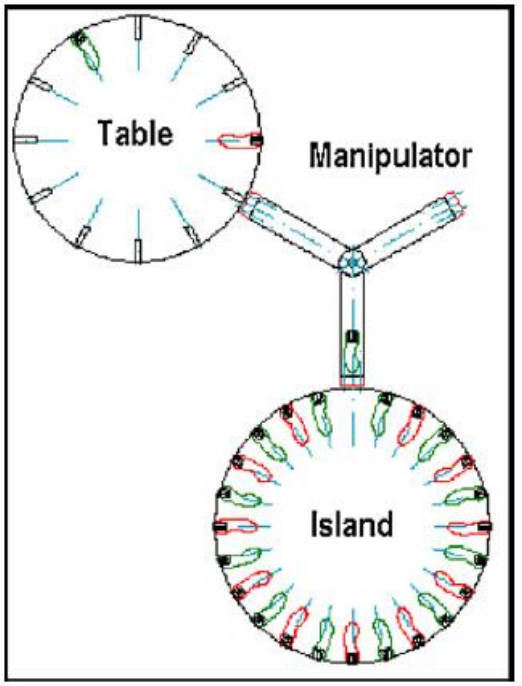

Fig. 5. Transportation tern.

Case System

- Manipulator module
  - The focus of the scheduling application refers to the manipulator module
  - 1)**Three pushers** enabling the **ejection of shoe molds and transferring** to the island or the table
  - 2)An **asynchronous motor driving** a globoidal cam precision mechanism enabling the **manipulator rotation**
  - 3)The proximity sensors ensuring the correct positioning of the manipulator
- control solution of the manipulator has a distributed architecture
  - **Three control modules** managing the three pushers
  - **A control module coordinating** the manipulator **rotation**
  - A **supervision module** ensuring the **synchronization** of the **manipulator** mechatronic devices (cylinders, motor and sensors) as well as the synchronization of the **manipulator with the island and the table of the tern**
- facts
  - testing application includes the coordination module(Rc) and the three pushers control modules(R1, R2, R3)
  - The pushers and the coordination control modules are physically executed on different controllers characterized by different cycle times(coordinator PLC cycle time is 100 ms, each pusher cycle time is 70 ms)
  - These four control modules have a total number of 29 automation tasks to be scheduled (coordinating module 14, each one of the pusher 5; 14+5*3=29; 29*6=174 total)

Experimental testing of the ISA approach

- experimental testing has been structured in two main stages
  - The first experimentation concerns a nominal scenario in which the control software is assumed to be correctly executed without any exogenous or endogenous unforeseen events
  - The second stage of the experimentation (not-nominal scenario) is realized in order to evaluate the behavior of ISA when unforeseen events occur

Experimental testing of the ISA approach

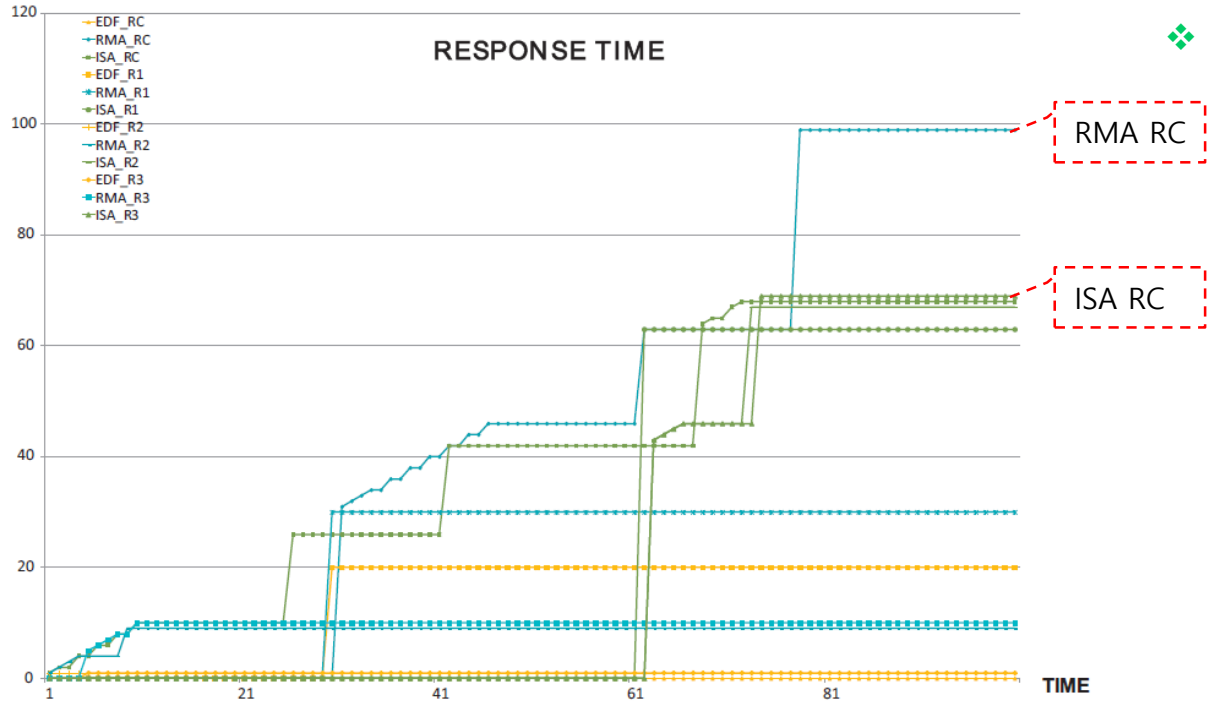

Fig. 6. Plot of response time in ms versus PLC cycle time in ms.

- ISA and RMA present comparable behavior (similar response time) **with regards to Rc up to time step 63**.
- After this time instant ISA globally performs much better by ensuring a response time up to **30% lower than RMA**
- In the **RMA algorithm** there is **no trace of this technological precedence** and it results that the **related** Response time values on average are **40% lower** than the response time resulting from ISA where they are taken into account in the problem formulation
  ➔ the modeling of a **high number of technological precedence between tasks** drastically **affects the task sequencing** and execution over time

Experimental testing of the ISA approach

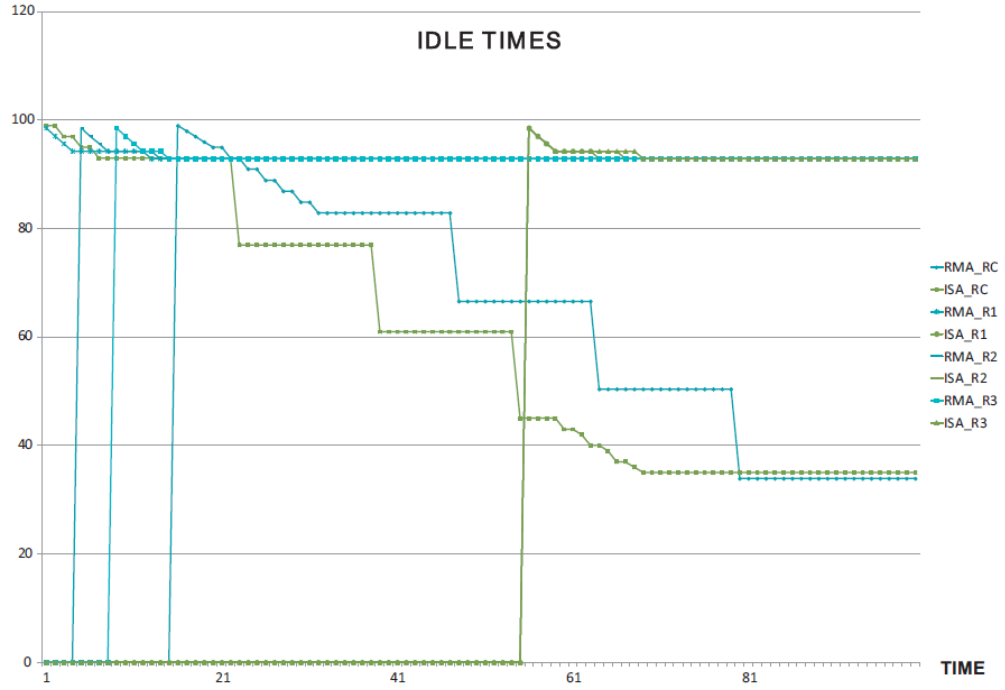

Fig. 7. Plot of idle time in ms versus PLC cycle time in ms.

- **EDF** performance is poor and thus **neglect**able in the comparison study
- it is possible to track the idle time progression for the ISA solution that **optimizes the task allocation in order to minimize the idle times** even taking into account the presence of technological constraints

Experimental testing of the ISA approach

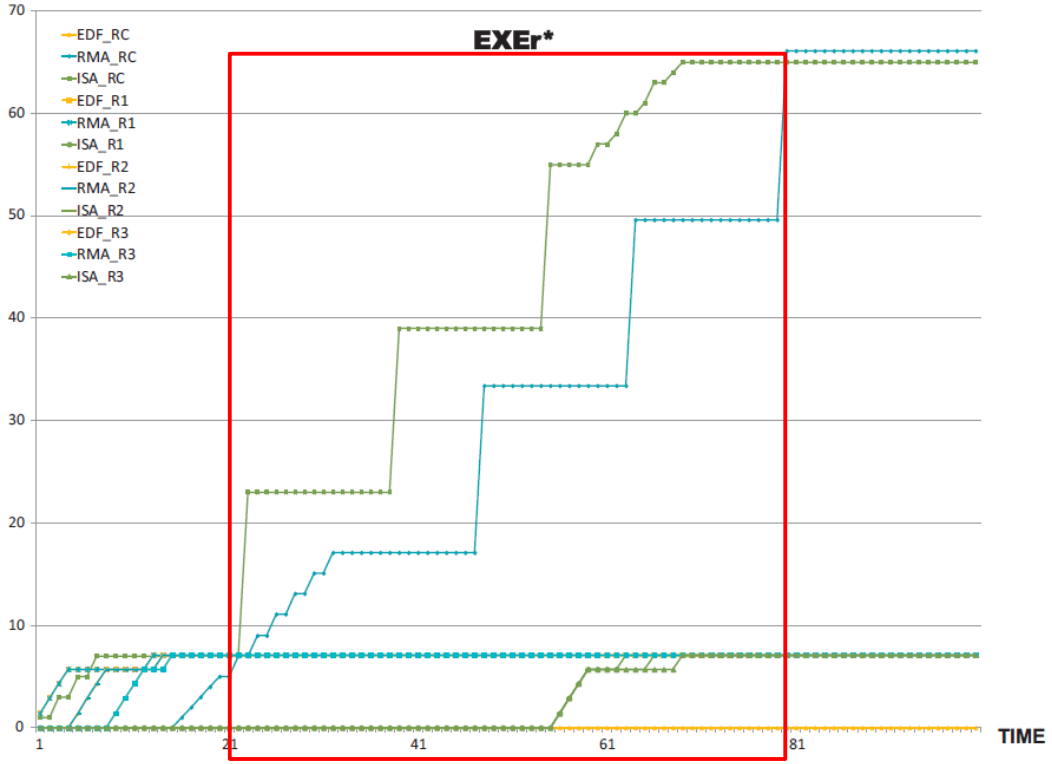

Fig. 8. Plot of *Exe** in ms versus PLC cycle time in ms.

- some differences with regards to the **Rc resource** that, **initially**, highlights a **utilization of 20% on average higher** by adopting the **ISA** approach compared to RMA whereas toward the **end** of the PLC cycle time both the curves tend to **reach a similar threshold**
- ISA adoption becomes smoother over time thanks to the existence of maximum processor utilization constraints

Experimental testing of the ISA approach

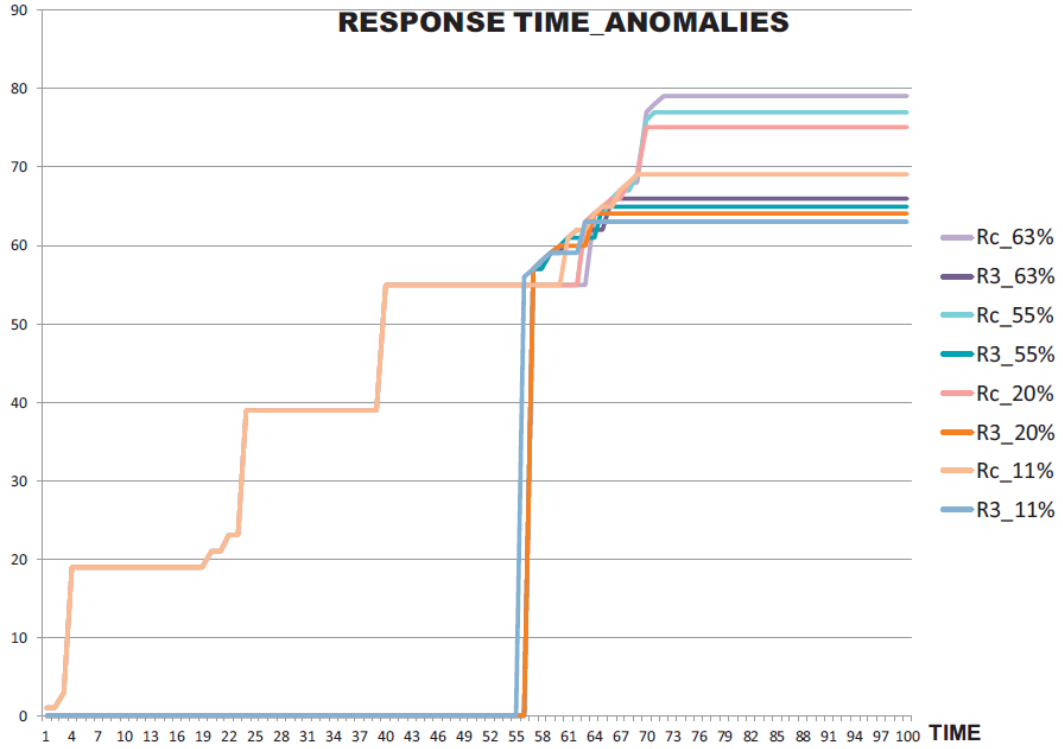

Fig. 9. Plot of response time with anomalies versus PLC cycle time in ms.

- Curves are plotted for all the percentage gaps only with regards to resources **Rc and R3** that are the most affected by the anomalies
- all tasks can be still scheduled by adopting ISA even with gaps of 63%
- The higher the gap, the more the value of the resources' response time increases
- Compared to ISA results, the same experimentation run with RMA and EDF led to a partial schedulability of the set of tasks

Experimental testing of the ISA approach

- The two Failure category
  - **Complete stop**
    - actual breakage of the resource
  - the manipulator is working and only **few devices are temporary blocked**
    - failures being realized while the manipulator is working and only few devices are temporary blocked

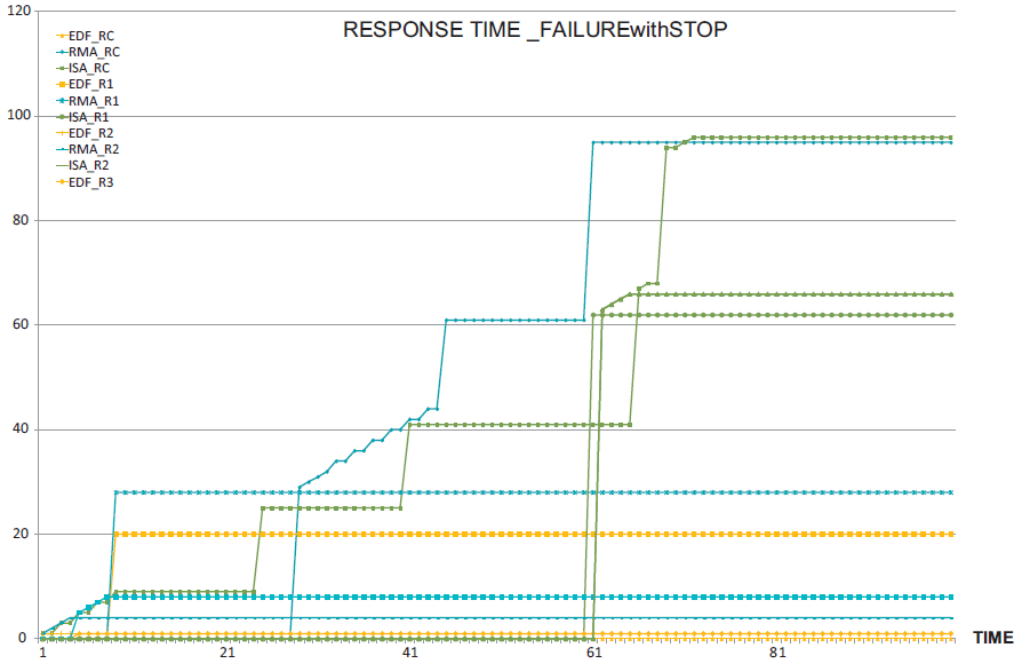

Fig. 10. Plot of response time in ms with failure and complete stop of the manipulator versus PLC cycle time in ms.

- **ISA enables the scheduling** of the tasks whereas **the other algorithms do not permit** the scheduling of respectively **one task for RMA and twelve tasks for EDF**
- ISA presents on average the best performance in terms of response times
- This behavior is **also** confirmed by the **Exe^*_r and idle time values**
- This is clearly **related to the extreme flexibility of ISA** in minimizing the idles versus RMA and EDF

Experimental testing of the ISA approach

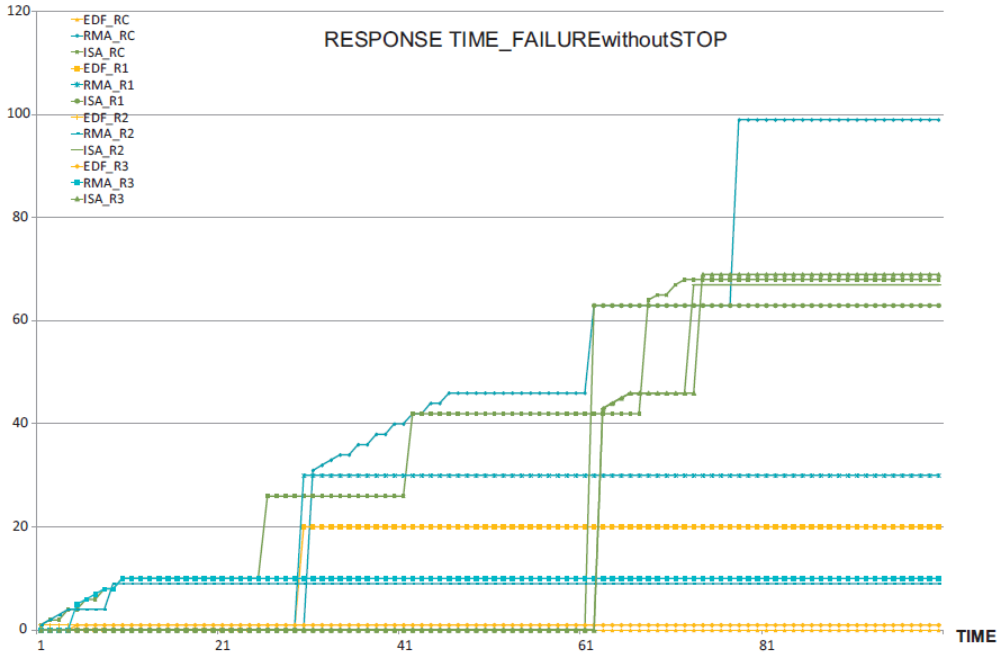

Fig. 11. Plot of response time in ms with failure and partial stop of the manipulator versus PLC cycle time in ms.

- In the case the failure of Pusher1 would have **not required the total stop** of the manipulator but only the inactivity of a subportion, **four tasks related to R1 would be aborted**.
- The damage would be handled by executing an extraordinary maintenance task specifically on R1 while preserving the ordinary maintenance tasks on R2 and R3.
- This would result in **a complete scheduling of the tasks by ISA** algorithm and **a partial scheduling by RMA and EDF**

Conclusion and future works

Conclusion and future works

- Compared to existing real-time scheduling algorithms, the ISA analytical formulation – consisting of 10 steps – enables the **modeling of technological precedence** between control tasks together with the **scheduling across multiple control resources** as well as an **efficient management of anomalies and failures** occurring during the control software execution.
- **ISA shows an enhanced capability to complete the scheduling plan** by allocating all the tasks, for all the experimental scenarios, even in cases of anomalies and severe failures
- ISA infrastructure is not constrained by any particular control solutions or software platform


감사합니다

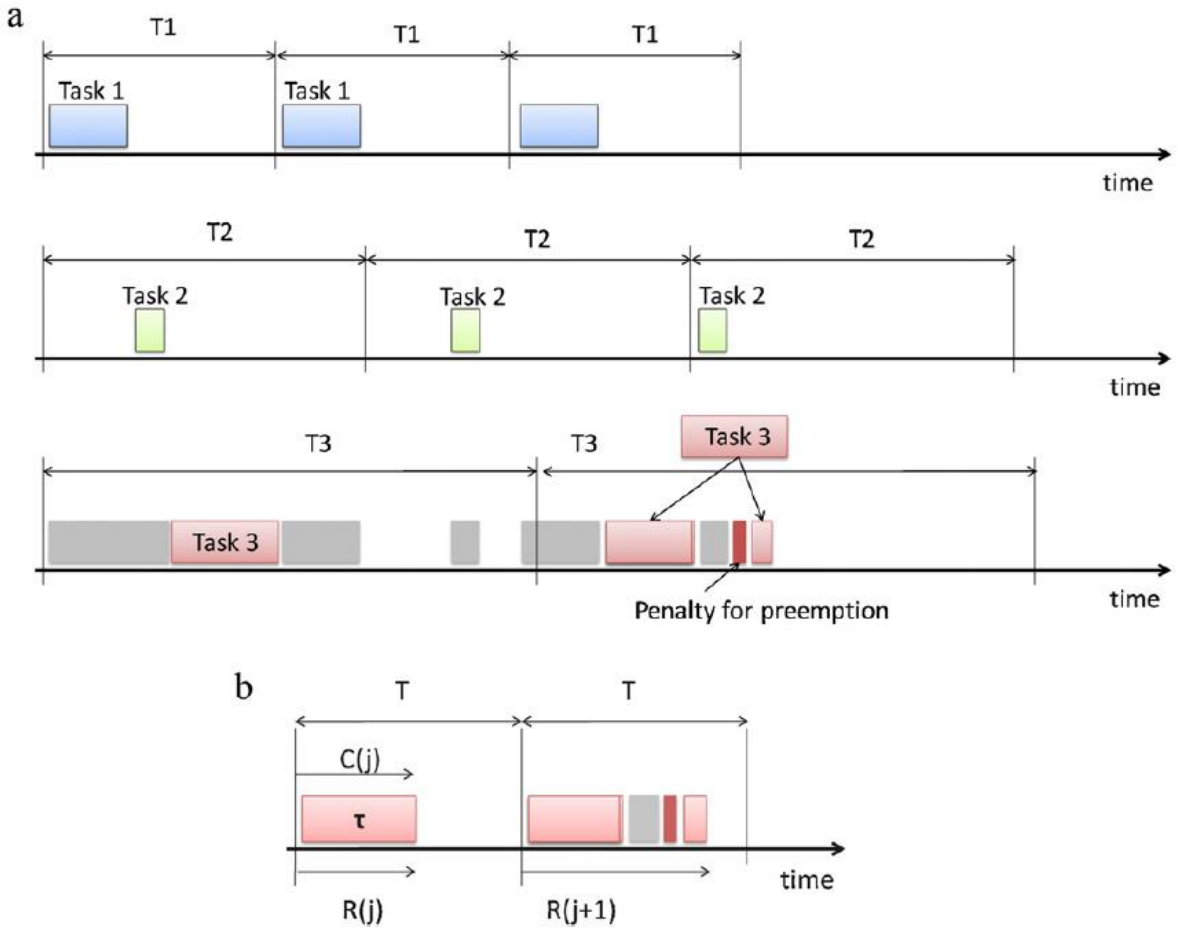

Fig. 1. (a) Periodic task preemption representation. (b) Periodic task preemption generalization.