Mixed-Criticality Real-Time Systems

Lecture #19

Anand Bhat

Outline

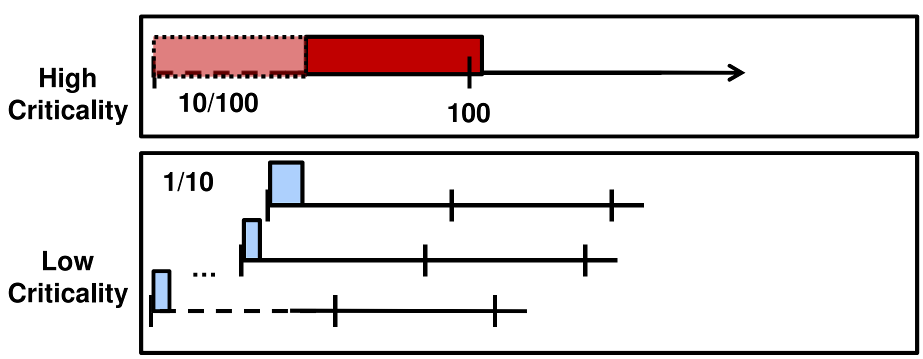
- Mixed-criticality real-time systems
- Zero-slack scheduling for **uni**processors
  - Zero-slack properties
- Generalizing resource allocation to **distributed** mixed-criticality tasks
  - Metric: Ductility matrix
- Compress-on-Overload Packing (COP)
  - COP Performance
- Radar surveillance case study
- Conclusions

Mixed-Criticality Real-Time Systems

- In traditional real-time systems, critical tasks were assigned to dedicated processors with non-critical tasks running on physically separate processors
  - Leads to inefficient use of resources
  - Requires additional processors, resulting in higher weight, volume and cooling requirements
- In "mixed-criticality real-time systems", tasks of different importance ("criticality") are co-located on the same resources
  - Goal: More efficient use of resources (i.e. higher utilization)
- Question: How to schedule these mixed-criticality tasks?
  - In dynamic environments where the execution-times can vary perhaps significantly

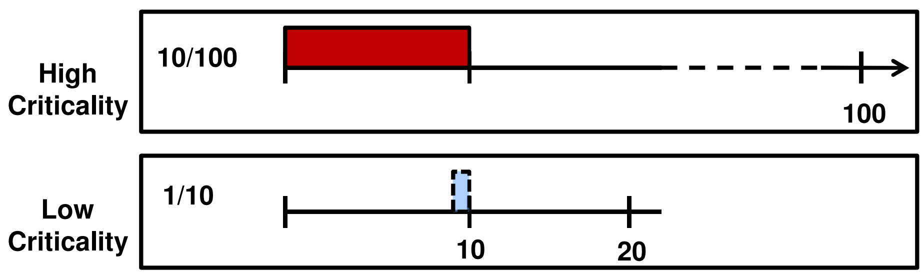
Rate-Monotonic Priority

- Shorter Period → Higher Priority
  - Ideal priority assignment
- BUT: Criticality Inversion
  - If criticality order is opposite to rate-monotonic priority order



Criticality As Priority Assignment (CAPA)

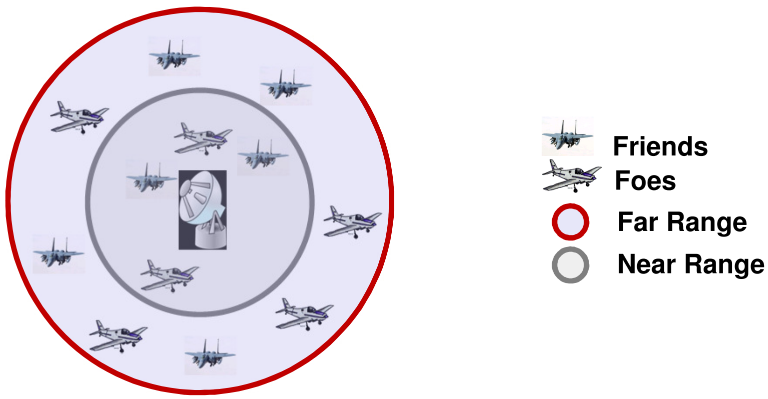
- Higher Criticality → Higher Priority
  - Ideal criticality protection:
    - lower criticality cannot interfere with higher criticality
- BUT: Poor Utilization Due to (RM) Priority Inversion
  - If criticality order is opposite to rate-monotonic priority order



Remember Period Transformation?

- How can we ensure the deadline of a critical task with a long period, resulting in a low priority?
  - "Period Transformation".
- For example, transform a task with period *T* and a worst-case execution time of *C* into a period-transformed task with a period of *T*/*k* and execution time *C*/*k* (where *k* = 2, 3, ….). Since the task period is now shorter, it can be assigned a higher priority.
  - importance and rate-monotonic priority assignment can be made consistent
- Optimal period transformation requires that each transformed (virtual) period have the same transformed value of "*C*"
  - This can be rather pessimistic in mixed-criticality systems where the worst-case value of C can vary very widely.

Radar Surveillance Example



Overload Situation

- Critical tasks may fail due to lack of resources
  - i.e. Important deadlines may be missed under overloads
- Desirable behavior
  - Under non-overloaded conditions, every task meets its deadlines
  - Under *overloaded* conditions, more critical tasks are favored over less critical tasks

Friends	Far Range	Low Criticality
Friends	Near Range	
Foes	Far Range	High Criticality
Foes	Near Range	

Illustration of Overload Behavior

- *A* and *B* are two resource allocation methods
- Consider an overload scenario *O*
  - Both *high* and *low* criticality tasks face an overload
- Under scheme *A*
  - Low criticality tasks meet their deadlines
  - High criticality tasks potentially miss deadlines
- Under scheme *B*
  - Low criticality tasks potentially miss deadlines
  - High criticality tasks meet their deadlines
- In mixed-criticality systems, ***"B is better than A"***

Task Model

$$\tau_i = \left(C_i, C_i^o, T_i, D_{i,} \zeta_i\right)$$

C_i **"Normal" Execution Budget of task τ_i**

C_i^o **"Overload" Execution Budget of task τ_i**

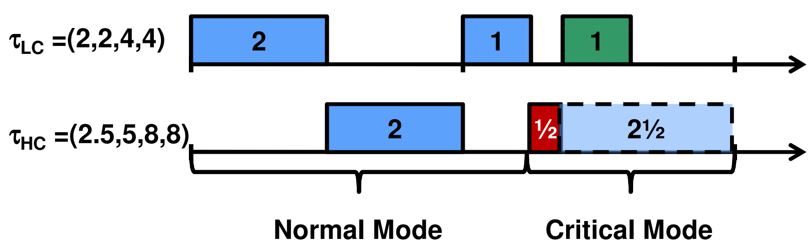
T_i **Period of task τ_i**

D_i **Deadline of task τ_i ($D_i \leq T_i$)**

ζ_i **Criticality of task *i* *(zeta-i)***

Zero-Slack Scheduling

- Start with RMS
- Calculate the last instant before τ_{HC} misses its deadline
  - this is called the ***zero-slack*** instant
- Switch to criticality-as-priority assignment at zero-slack instant
  - Splits the execution window into
    - **Normal mode** (RM)
    - **Critical mode** (CAPA)



Interference in Zero-Slack Scheduling

- Task set divided into
  - H^{lc} : Higher priority, lower criticality
  - H^{hc}: Higher priority, higher criticality
  - L^{lc}: Lower priority, lower criticality
  - L^{hc}: Lower priority, higher criticality
- Interfering tasks in normal mode (Normal mode)
  - $H^{lc} + H^{hc} + L^{hc}$
- Interfering tasks in critical mode (Critical mode)
  - $H^{hc} + L^{hc}$

Interference → preemption from a task due to its ***higher priority in normal mode*** or ***higher criticality in critical mode***

Generic Scheduling Algorithm

Slack calculation depends on priority-based scheduler

Schedulability

$\forall i\ Z_i^1 \Leftarrow 0$
repeat
  $\forall i\ Z_i^0 \Leftarrow Z_i^1$
  for all i in taskset **do**
    $V_i^n \Leftarrow GetSlackVector(i, \Gamma_i^n)$
    $V_i^c \Leftarrow GetSlackVector(i, \Gamma_i^c)$
    $Z_i^1 \Leftarrow GetSlackZeroInstant(i, V_i^c, V_i^n, t)$
  end for
until $\forall i\ Z_i^0 = Z_i^1$
return Z_i^1

GetSlackZeroInstant(i, V^c,V^n,t)

$C_i^c \Leftarrow C_i^o$; $C_i^n \Leftarrow 0$
repeat
  $t_1 \Leftarrow StartOfTrailingSlack(i, C_i^c, V^c)$
  if $t_1 \geq 0$ and $t_1 \leq t$ **then**
    $k_u \Leftarrow SlackUpToInstant(V^n, t_1) - C_i^n$
    $k_u = \max(\min(k_u, C_i^c), 0)$
    $C_i^c \Leftarrow C_i^c - k_u$
    $C_i^n \Leftarrow C_i^n + k_u$
  else
    $k_u \Leftarrow 0$
  end if
until $k_u = 0$
return t_1

Zero-Slack RM (ZSRM) Properties

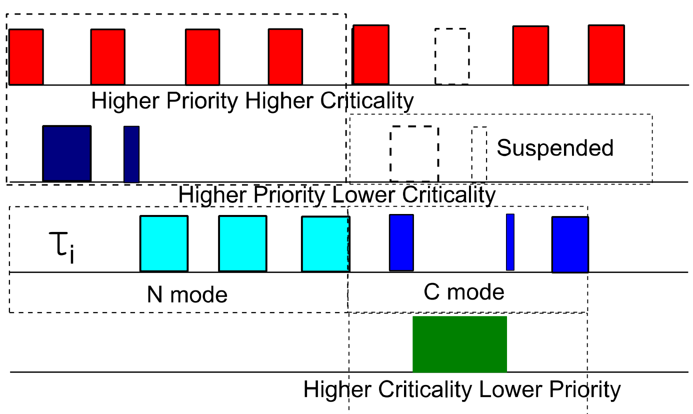
- Subsumes RM
  - If criticalities are aligned with priorities
  - No *critical mode*
- Subsumes CAPA
  - If not enough slack, *only critical mode*
- Graceful Degradation
  - In overloads, deadlines are missed in inverse criticality order

Scheduling Guarantee

A task τ_i is guaranteed C_i^o before D_i

if no τ_j with higher criticality than τ_i executes beyond its C_j

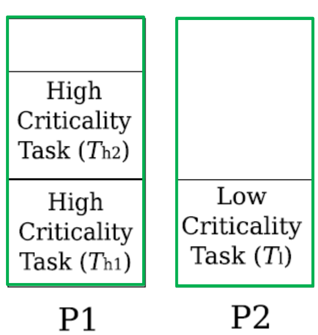
Critical Instant of a Task τ_i



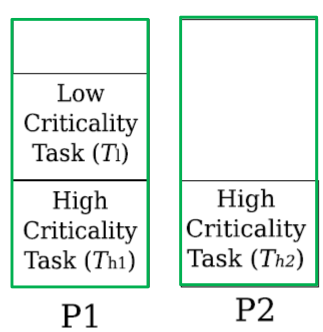
MULTIPROCESSOR OR DISTRIBUTED SYSTEMS

Example Packing

$Task$	C_i (ms)	C_i^o (ms)	$Period$ T_i (ms)	$Criticality$ $\kappa(\tau_i)$
τ_{h1}	4	6	10	1
τ_{h2}	4	6	10	1
τ_l	2	3	5	2



Packing that incurs criticality inversion



Packing that avoids criticality inversion

Measuring Allocation Effectiveness

Example with 2 tasks:

One with high criticality and the other with low criticality

← Criticality

$$
\begin{array}{l|cc}
 & \text{High } (k_1) & \text{Low } (k_2) \\
\hline
W_1\text{: Both-Criticality Overload} & d_{1,1} & d_{1,2} \\
W_2\text{: High-Criticality Overload} & d_{2,1} & d_{2,2} \\
W_3\text{: Low-Criticality Overload} & d_{3,1} & d_{3,2} \\
W_4\text{: No Overload} & d_{4,1} & d_{4,2}
\end{array}
$$

Overload ↓

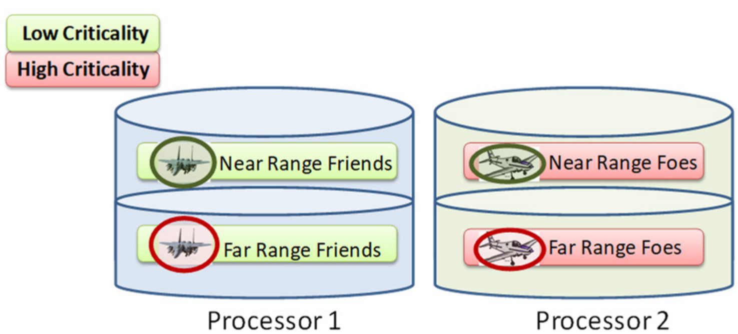
Let $d_{i,j}$ indicate whether tasks in criticality level-*i* meet their deadlines under overload W_j

$d_{i,j}=1$, indicates that tasks in criticality level-*i* meet their deadlines under overload W_j

$d_{i,j}=0$, indicates that tasks in criticality level-*i* **do not** meet their deadlines under overload W_j

***i* corresponds to column, *j* corresponds to row**

Criticality Isolation Strategy (S)

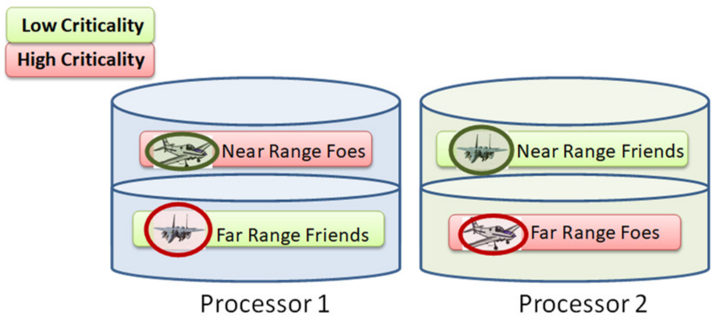


	H	L
Both overload	0	0
High criticality overload	0	1
Low criticality overload	1	0
No overload	1	1

Assume that the system is schedulable without overloads

Under overloads only one task can meet its deadline

Criticality Mixture Strategy (T)



	H	L
Both overload	1	0
High criticality overload	1	0
Low criticality overload	1	0
No overload	1	1

T is better than S

Assume that the system
is schedulable without overloads
Under overloads only one
task can meet its deadline
Assume that a uniprocessor
mixed-criticality scheduling algorithm
like ZSRM is used within each processor

Generalization: Ductility Matrix

$$
\begin{array}{cc}
 & \xleftarrow{\quad} \text{Criticality} \\
\text{Overload} \downarrow & \begin{pmatrix} d_{1,1} & d_{1,2} & d_{1,3} & \dots & d_{1,k} \\ d_{2,1} & d_{2,2} & d_{2,3} & \dots & d_{2,k} \\ \dots & & & \dots & \\ d_{2^k,1} & d_{2^k,2} & d_{2^k,3} & \dots & d_{2^k,k} \end{pmatrix}
\end{array}
$$

Say we have ‘k’ criticality levels
2^k possible overload scenarios
- *All criticality levels overload* to *No overload*

Quantification of Ductility

$$P_d(D) = \sum_{c=1}^{k} \left\{ \frac{1}{2^c} \frac{\sum_{r=1}^{2^k} d_{r,c}}{2^k} \right\}$$

*Scheme **S***

	H	L
Both overload	0	0
High criticality overload	0	1
Low criticality overload	1	0
No overload	1	1

*Scheme **T***

	H	L
Both overload	1	0
High criticality overload	1	0
Low criticality overload	1	0
No overload	1	1

For S, P_d (D) = 0.375

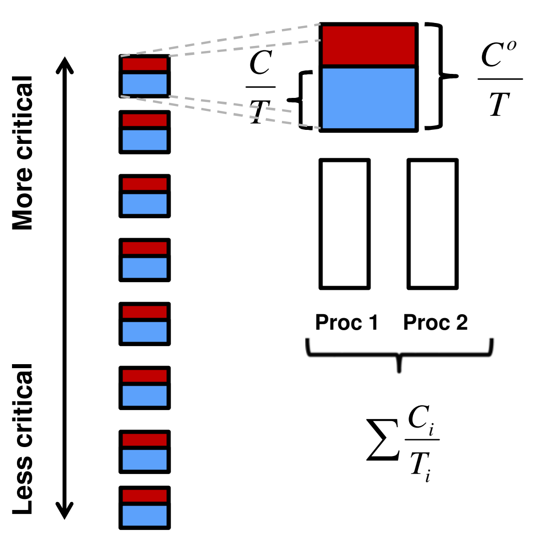
For T, P_d (D) = 0.5625

Shows that ***T*** is better than ***S***

Other Projection functions can be used

P_d(D) favors the more critical tasks **exponentially** over the lower criticality tasks

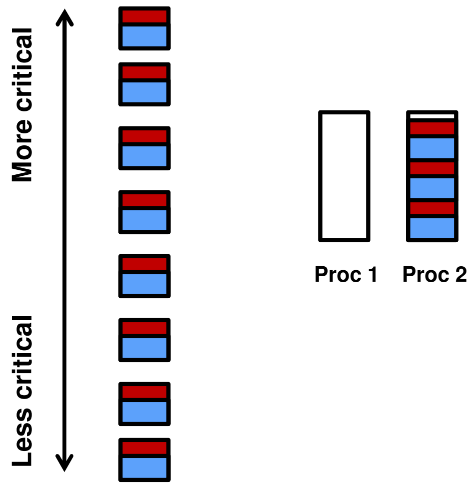
Compress-on-Overload Packing (COP)



Compress-on-Overload Packing (COP)

Phase 1: Pack by criticality, then size

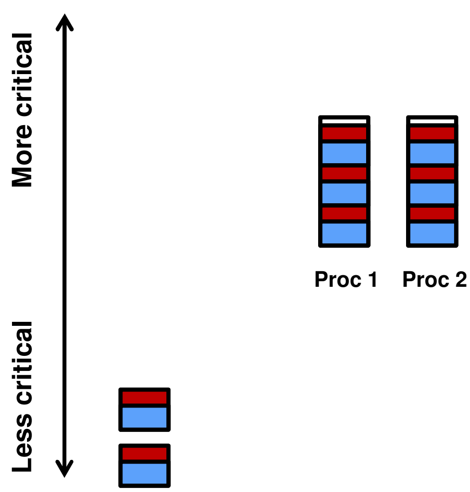
object size = $\dfrac{C_i^o}{T_i}$



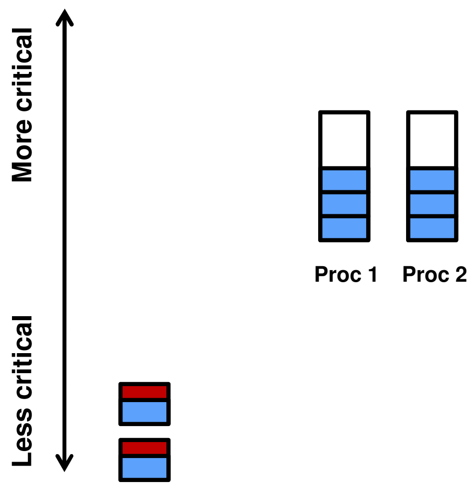
Compress-on-Overload Packing (COP)

Phase 2: Pack by criticality, then size

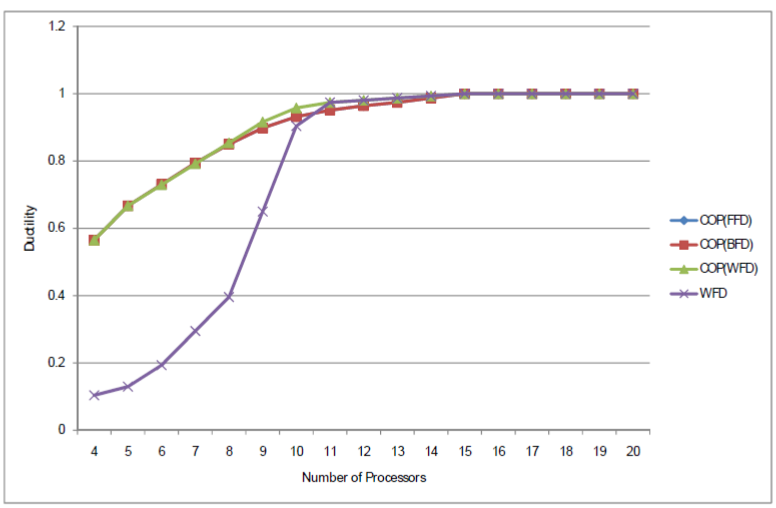
object size = $\dfrac{C_i}{T_i}$



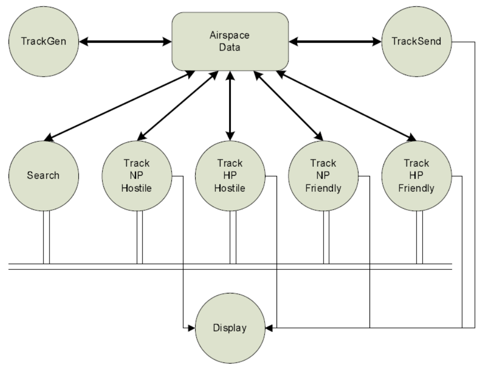
Compress-on-Overload Packing (COP)



COP Performance



Radar Surveillance Simulation Case Study



HP: High Priority

NP: Normal Priority

Platform: Intel Core 2 Extreme 2.526GHz 32KB I-Cache, 32KB D-Cache,6MB L2 Cache

Task	C_i (ms)	C_i^o (ms)	*Period* T_i (ms)	*Criticality* $\kappa(\tau_i)$
HP Hostile	40	58	100	1
NP Hostile	83	106	200	1
HP Friendly	40	58	100	2
NP Friendly	83	106	200	2

Radar Surveillance Deadline Misses

Packer	Deadline misses (%) / Task missing deadlines		
	2300 Tracks	2400 Tracks	2500 Tracks
WFD	0	24.32/NP Hostile	69.56 / NP Hostile
COP	0	10.34/HP Friendly	59.18/ HP Friendly

$$Ductility(WFD) = \begin{array}{r} w=3=<1,1> \\ w=2=<1,0> \\ w=1=<0,1> \\ w=0=<0,0> \end{array} \begin{pmatrix} 00 \\ 01 \\ 10 \\ 11 \end{pmatrix} = \left(\frac{1}{2}\frac{2}{4}+\frac{1}{4}\frac{2}{4}\right) = 0.375 \qquad normalized = \frac{0.375}{0.750} = 0.5$$

$$Ductility(COP) = \begin{array}{r} w=3=<1,1> \\ w=2=<1,0> \\ w=1=<0,1> \\ w=0=<0,0> \end{array} \begin{pmatrix} 10 \\ 11 \\ 11 \\ 11 \end{pmatrix} = \left(\frac{1}{2}\frac{4}{4}+\frac{1}{4}\frac{3}{4}\right) = 0.675 \qquad normalized = \frac{0.6875}{0.750} = 0.9167$$

Conclusions

- Mixed-Criticality tasksets require temporal protection
  - Criticality Inversion with RM priority assignments
- Zero-Slack scheduler (meta-Scheduler) for Priority-Based Schedulers
  - Ensures criticality inversion does not lead to deadline misses
- New scheduling guarantee
  - τ_i is guaranteed C_i^o if all higher-criticality tasks τ_j consume C_j or less
- Compress-on-Overload Packing algorithm
  - Gives more overload room to critical tasks
- Ductility metric to measure effectiveness of allocation
- Radar case study demonstrates utility