Scheduling Mixed-Criticality Real-Time Systems

Doctoral Dissertation Defense
Under the Direction of Prof. Sanjoy K. Baruah
Outline

- Motivation
- Thesis statement
- Background
- Dissertation research
  - Models
  - Algorithms
- Other contributions and future work
Outline

- Motivation
- Thesis statement
- Background
- Dissertation research
  - Models
  - Algorithms
- Other contributions and future work
Cyber-Physical: The Next Revolution?

- Classic computer systems
  - Process information

- Cyber-physical systems
  - Interact with physical world
    - Collect data
    - Make decisions intelligently
    - Perform actions
Real-Time: Essential to Cyber-Physical

- Not only the right move

- The right move at a good time
Real-Time Systems

- Systems that provide both logical and temporal correctness
  - Predictability is more important than performance
  - Provably guarantee responses within strict time constraints (deadlines)
Real-Time == Real-Fast?

- Not true for multitasking systems

  - A good scheduling policy is the key to prompt responses
Motivation: Mixed-Criticality Systems

- Northrop Grumman X-47B
  - Unmanned aerial vehicle
    - Motion plan, route plan, recon and combat
    - Integrated, intelligent and computational-intensive
  - The flight control tasks are safety-critical
  - The mission-related tasks are non-safety-critical
Motivation: Mixed-Criticality Systems

- Conflicting requirements
  - Safety-critical tasks may never fail
    - Reserving large amount of resources
  - Non-critical tasks should be intelligent
    - Using as many resources as possible

- Traditional real-time systems face challenges
  - Computational resource waste due to pessimism
Motivation: Mixed-Criticality Systems

- Pessimism in traditional methods
  - Scheduling based on execution-time estimations
  - Safety-critical tasks require very pessimistic execution-time estimations
  - Non-critical tasks suffer from pessimistic estimations

Non-critical task

Safety-critical task

Impossible to get real-time response
Motivation: Mixed-Criticality Systems

- Pessimism in traditional methods
  - Non-critical tasks don’t have to assume such pessimistic estimations
    - Can assume smaller estimations and make guarantees
    - These guarantees are invalid only in the very worst case

![Diagram showing scheduling of mixed-criticality tasks](image)

- Safety-critical task
- Non-critical task

In most of the cases, only this amount needed
Can get real-time response (in most of the cases)
Motivation: Mixed-Criticality Systems

- Provide correctness guarantees in two levels
  - Large estimations and high-criticality constraints
  - Small estimations and low-criticality constraints

- What scheduling algorithms shall we use?
- Can we reduce resource waste?
Outline

- Motivation
- Thesis statement
- Background
- Dissertation research
  - Models
  - Algorithms
- Other contributions and future work
Thesis Statement

- New methods can be discovered to schedule real-time systems with multiple criticalities
- The methods can supply multiple temporal predictability assertions with respect to multiple WCET specifications
- The assertions can be defined and measured through a formalized description
- The methods can be efficiently implemented with acceptable computational complexities
Outline

- Motivation
- Thesis statement
- Background
- Dissertation research
  - Models
  - Solutions
- Other contributions and future work
The abstract model for real-time systems

- A one-shot Job is released at its release time, takes at most its worst-case execution time to finish, and is required to be finished by its deadline.
Real-Time Scheduling Theory

- The abstract model for real-time systems
  - Sporadic tasks recurrently release jobs
    - **Period**: the *minimal* gap between job releases

\[ \tau = (3,1) \]
Real-Time Scheduling Theory

- The abstract model for real-time systems
  - Sporadic tasks recurrently release jobs
    - **Relative deadline**: the gap between a job’s release time and its deadline
      - Implicit deadline: relative deadline is equal to period
        - Known as *Liu & Layland tasks*

```
<table>
<thead>
<tr>
<th>Job deadlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>τ=(3,1)</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Relative Deadline = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
```

Scheduling Mixed-Criticality Real-Time Systems
**Haohan Li**, Doctoral Dissertation Defense
April 26th, 2013, Chapel Hill
Real-Time Scheduling Theory

- The abstract model for real-time systems
  - Sporadic tasks recurrently release jobs
    - Relative deadline: the gap between a job’s release time and its deadline
      - Implicit deadline: relative deadline is equal to period
      - Arbitrary deadline: relative deadline isn’t equal to period

\[ \tau = (3, 2, 1) \]
Real-Time Scheduling Theory

- The abstract model for real-time systems
  - One-shot jobs
    - Release-time, deadline, worst-case execution time
  - Sporadic tasks
    - Period, relative deadline, worst-case execution time
  - Utilization and load to measure CPU capacity
    - Utilization: the overall fraction of processor time demand of a system
    - Load: the maximum fraction of processor time demand of a system over any time interval
Real-Time Scheduling Theory

- The abstract model for real-time systems

  - CPU capacity measurement: load and utilization

  - One-shot job
    - J=(0,6,5)
    - Load = 5/6
    - Utilization is undefined

  - Arbitrary-deadline task
    - \( \tau_1=(3,2,1) \)
    - Load = 1/2
    - Utilization = 1/3

  - Implicit-deadline task
    - \( \tau_2=(3, 1) \)
    - Load = Utilization = 1/3
Outline

- Motivation
- Thesis statement
- Background
- Dissertation research
  - Models
  - Algorithms
- Other contributions and future work
Dissertation Research

- New model for real-time systems
  - Mixed-criticality systems
    - Additional parameters and constraints
- Scheduling algorithms in the new model
  - OCBP algorithm
  - EDF-VD algorithm
- Capacity loss and optimality analysis
Contributions

- First solutions to many fundamental questions
  - One-shot jobs
  - Sporadic arbitrary-deadline tasks
  - Multiprocessor scheduling
- Effectively reduce resource waste
  - Both algorithms are proved to utilize resources more efficiently than traditional methods
  - Both algorithms have the best *speedup factor* for two criticality levels
Outline

- Motivation
- Thesis statement
- Background
- Dissertation research
  - Models
  - Algorithms
- Other contributions and future work
Model of Mixed-Criticality Systems

- Parameters for mixed-criticality tasks
  - Inherited from traditional real-time systems
  - Exclusive for mixed-criticality systems

<table>
<thead>
<tr>
<th>Period</th>
<th>WCETs</th>
<th>Deadline</th>
<th>Criticality</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td></td>
<td></td>
<td>High</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td></td>
<td></td>
<td>Mid</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td></td>
<td></td>
<td>Low</td>
</tr>
</tbody>
</table>
Model of Mixed-Criticality Systems

- Parameters for mixed-criticality tasks
  - Inherited from traditional real-time systems
  - Exclusive for mixed-criticality systems

\[ \tau_1 = (5, 4, [1, 2, 4], \text{High}) \]
\[ \tau_2 = (5, 4, [1, 2, -], \text{Mid}) \]
\[ \tau_3 = (5, 3, [1, -], \text{Low}) \]
Model of Mixed-Criticality Systems

- Valid scheduling algorithms
  
  - Basic idea: if high-criticality tasks overruns, low-criticality tasks can be ignored
Model of Mixed-Criticality Systems

- Valid scheduling algorithms
  - If no task exceeds level-X WCET, all tasks with level-X and above should meet their deadlines.

\[ \tau_1 = (5,4,[1,2,4],\text{High}) \]

\[ \tau_2 = (5,4,[1,2,-],\text{Mid}) \]

\[ \tau_3 = (5,3,[1,-,-],\text{Low}) \]
Valid scheduling algorithms

- If no task exceeds level-X WCET, all tasks with level-X and above should meet their deadlines.

\[
\begin{align*}
\tau_1 &= (5, 4, [1, 2, 4], \text{High}) \\
\tau_2 &= (5, 4, [1, 2, -], \text{Mid}) \\
\tau_3 &= (5, 3, [1, -,-], \text{Low})
\end{align*}
\]
Model of Mixed-Criticality Systems

- Valid scheduling algorithms
  - If no task exceeds level-X WCET, all tasks with level-X and above should meet their deadlines
Outline

- Motivation
- Thesis statement
- Background
- Dissertation research
  - Models
  - Algorithms
- Other contributions and future work
The challenge is that we do not know if a high-level task will use the high-level WCET

- We denote this as *non-clairvoyant*
- The scheduling policy must detect the behavior of the system, and drop tasks when necessary
Scheduling Mixed-Criticality Systems

Scheduling Mixed-Criticality Real-Time Systems

Haohan Li, Doctoral Dissertation Defense
April 26th, 2013, Chapel Hill
Scheduling Mixed-Criticality Systems

- The challenge is that we do not know if a high-level task will use the high-level WCET
  - The scheduling policy must detect the behavior of the system, and drop tasks when necessary
  - When a low-criticality and urgent job and a high-criticality and non-urgent job are both pending, we have to make a proper decision
Example: Mixed-Criticality Jobs

- Here is a detailed example to explain our solutions to the questions within our model
  - One-shot jobs
  - Two criticality levels

\[ J_1 = (0, 3, [2,-], \text{Low}) \]
\[ J_2 = (0, 6, [2,-], \text{Low}) \]
\[ J_3 = (0, 8, [1,3], \text{High}) \]
\[ J_4 = (0, 9, [1,3], \text{High}) \]
Example: Mixed-Criticality Jobs

- Traditional scheduling policies don’t work efficiently on mixed-criticality systems
  - Traditional method: Earliest-deadline-first (EDF)
  - Job scheduling priorities: $J_1 < J_2 < J_3 < J_4$

$J_1 = (0, 3, [2, -], \text{Low})$
$J_2 = (0, 6, [2, -], \text{Low})$
$J_3 = (0, 8, [1, 3], \text{High})$
$J_4 = (0, 9, [1, 3], \text{High})$
Example: Mixed-Criticality Jobs

- Traditional scheduling policies don’t work efficiently on mixed-criticality systems
  - Traditional method: Criticality-monotonic
  - Job scheduling priorities: $J_3 < J_4 < J_1 < J_2$

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrival</th>
<th>Departure</th>
<th>Criticality</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>(0,3)</td>
<td>(2, -)</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>$J_2$</td>
<td>(0,6)</td>
<td>(2, -)</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>$J_3$</td>
<td>(0,8)</td>
<td>(1,3)</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>$J_4$</td>
<td>(0,9)</td>
<td>(1,3)</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>
Scheduling Mixed-Criticality Systems

- Traditional scheduling policies don’t work efficiently on mixed-criticality systems
  - Urgency and importance may differ

![Diagram showing scheduling examples]

- \( J_1 = (0,3,[2,-],\text{Low}) \)
- \( J_2 = (0,6,[2,-],\text{Low}) \)
- \( J_3 = (0,8,[1,3],\text{High}) \)
- \( J_4 = (0,9,[1,3],\text{High}) \)

RM/EDF favors only urgent jobs

Criticality monotonic favors only important jobs
Evaluation of Scheduling Algorithms

- Mixed-criticality scheduling problem is NP-hard in the strong sense
  - The performance of scheduling algorithms is quantified in the form of speedup factors
    - An algorithm has a speedup factor $s$ if it can schedule a task set that is schedulable by a clairvoyant scheduling algorithm on a speed-$s$ processor
      - Exact algorithm has a speedup factor of 1, but no polynomial or pseudo-polynomial algorithm can reach it
      - Smaller factor represents better performance
OCBP Algorithm

- The first solution: OCBP algorithm [RTAS 10]
  - Own-criticality-based-priority algorithm
    - An approximation algorithm for one-shot jobs in preemptive uniprocessor systems
    - Priorities are assigned to jobs before run-time
    - OCBP seeks a balance between urgency and importance

OCBP Algorithm

- The first solution: OCBP algorithm [RTAS 10]
  - Own-criticality-based-priority algorithm
    - An approximation algorithm for one-shot jobs in preemptive uniprocessor systems

J₁=(0,3,[2,-],Low)
J₂=(0,6,[2,-],Low)
J₃=(0,8,[1,3],High)
J₄=(0,9,[1,3],High)

Time
OCBP Algorithm

- OCBP algorithm constructs a priority list
  - In this example, priority list is $J_1 < J_3 < J_4 < J_2$
  - Basic idea: a job should ignore jobs with lower priorities, and tolerate jobs with higher priorities

<table>
<thead>
<tr>
<th>Job</th>
<th>Start</th>
<th>Duration</th>
<th>Release</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>0</td>
<td>3</td>
<td>-</td>
<td>Low</td>
</tr>
<tr>
<td>$J_2$</td>
<td>0</td>
<td>6</td>
<td>-</td>
<td>Low</td>
</tr>
<tr>
<td>$J_3$</td>
<td>0</td>
<td>8</td>
<td>[1,3]</td>
<td>High</td>
</tr>
<tr>
<td>$J_4$</td>
<td>0</td>
<td>9</td>
<td>[1,3]</td>
<td>High</td>
</tr>
</tbody>
</table>
OCBP Algorithm

- OCBP algorithm constructs a priority list
  - In this example, priority list is $J_1 < J_3 < J_4 < J_2$
  - Basic idea: a job should ignore jobs with lower priorities, and tolerate jobs with higher priorities

$J_1 = (0, 3, [2, -], \text{Low})$

$J_2 = (0, 6, [2, -], \text{Low})$

$J_3 = (0, 8, [1, 3], \text{High})$

$J_4 = (0, 9, [1, 3], \text{High})$

When $J_2$ get the lowest priority, it will not fail

If $J_3$ or $J_4$ overrun, $J_2$ can validly miss its deadline
OCBP Algorithm

- OCBP algorithm constructs a priority list
  - In this example, priority list is $J_1 < J_3 < J_4 < J_2$
  - Basic idea: a job should ignore jobs with lower priorities, and tolerate jobs with higher priorities

$J_1 = (0, 3, [2, -], \text{Low})$
$J_2 = (0, 6, [2, -], \text{Low})$
$J_3 = (0, 8, [1, 3], \text{High})$
$J_4 = (0, 9, [1, 3], \text{High})$

J_4 will not fail even if J_1 and J_3 run to max WCET
J_2 won’t affect J_4 – it has a lower priority

Scheduling Mixed-Criticality Real-Time Systems
Haohan Li, Doctoral Dissertation Defense
April 26th, 2013, Chapel Hill
OCBP Algorithm

- OCBP algorithm constructs a priority list
  - In this example, priority list is $J_1 < J_3 < J_4 < J_2$
  - Basic idea: a job should ignore jobs with lower priorities, and tolerate jobs with higher priorities
OCBP Algorithm

- OCBP algorithm constructs a priority list
  - The algorithm will recursively seek the lowest-priority job, and build the list from backward

J₁=(0,3,[2,-],Low)
J₂=(0,6,[2,-],Low)
J₃=(0,8,[1,3],High)
J₄=(0,9,[1,3],High)
Evaluation of OCBP Algorithm

- Speedup factor [RTAS 10]
  - Speedup factor of EDF for two criticality levels is 2
    - Intuitively, EDF requires a full processor for each criticality level, which sums up to 2
  - Speedup factor of OCBP algorithm for two criticality levels is 1.618
    - The factor 1.618 is the golden ratio
Evaluation of OCBP Algorithm

- **Speedup factor [TC 12]**
  - OCBP can handle more than two criticality levels
  - Speedup factor of EDF for N criticality levels is N
    - Still, EDF requires a full processor for each criticality level, which sums up to N
  - Speedup factor of OCBP for N criticality levels is the root of the equation \((x+1)^{N-1} = x^N\)
    - The asymptotic form is \(\Theta(N/\log N)\)

Evaluation of OCBP Algorithm

- Speedup factor [TC 12]
  - No scheduling algorithms can have a speedup factor better than 1.618 for two criticality levels
    - OCBP algorithm is optimal with respect to speedup factors for two criticality levels
  - No fixed-job-priority scheduling algorithms can have a speedup factor better than the root of the equation $(x+1)^{N-1} = x^N$
EDF-VD Algorithm

- OCBP algorithm works on one-shot jobs
  - Extending to sporadic tasks is complicated
    - Priorities must be assigned to each job

- EDF-VD algorithm on sporadic tasks [ECRTS 12]
  - Run-time complexity is significantly improved

EDF-VD Algorithm

- OCBP algorithm works on one-shot jobs
  - Extending to sporadic tasks is complicated
    - Priorities must be assigned to each job

- EDF-VD algorithm on sporadic tasks [ECRTS 12]
  - Earliest-Deadline-First with Virtual Deadlines
  - Based on EDF – the optimal scheduling algorithm on traditional real-time systems
  - The priority of each job is simply determined according to its virtual deadline
EDF-VD Algorithm

- **Basic idea**
  - Give high-criticality tasks earlier (and virtual) deadlines
    - Not only urgent jobs are favored
    - High-criticality tasks will also get higher priorities

- **Base case**
  - Two criticality levels
  - Firstly on implicit-deadline tasks, then on arbitrary-deadline tasks
EDF-VD Algorithm

- Virtual deadlines are assigned proportionally to original deadlines with factor $x \ (0<x<1)$
  - Use virtual deadlines in low-criticality behavior
  - Change to original deadlines when high-criticality behavior is detected
- My research proposes a mechanism to choose $x$
- My research proposes a schedulability test

- The analysis of EDF-VD is different from OCBP
  - Priorities are relative; deadlines are absolute
EDF-VD Algorithm

- In the example, we try to set a scaling factor $x=\frac{1}{2}$ for high-criticality tasks
  - $\tau_2$ and $\tau_3$ use virtual deadlines if no criticality change is detected
  - Will any job miss its deadline?

$\tau_1=(2,[1,-],Low)$
$\tau_2=(8,[1,3],High)$
$\tau_3=(9,[1,3],High)$
EDF-VD Schedulability Test

- **Basic idea**
  - Check the worst case response time for any job
  - Try to find as many jobs as possible that have earlier deadlines and can preempt the candidate job

\[ \tau_1 = (2, [1, -], \text{Low}) \]
\[ \tau_2 = (8, [1, 3], \text{High}) \]
\[ \tau_3 = (9, [1, 3], \text{High}) \]
EDF-VD Schedulability Test

- In the **low-criticality** behavior, which jobs can have earlier deadlines and preempt the job?
  - In the example, we check the 2\textsuperscript{nd} job of $\tau_1$

![Diagram](image-url)

- $\tau_1=(2,[1,-],\text{Low})$
- $\tau_2=(8,[1,3],\text{High})$
- $\tau_3=(9,[1,3],\text{High})$

**Low-level jobs with earlier deadline**

**High-level jobs with earlier virtual deadline**
EDF-VD Schedulability Test

- In the **high-criticality** behavior, which jobs can have earlier deadlines and preempt the job?
  - In the example, we check the 1st job of $\tau_2$

- $\tau_1 = (2, [1,-], \text{Low})$
- $\tau_2 = (8, [1,3], \text{High})$
- $\tau_3 = (9, [1,3], \text{High})$
EDF-VD Schedulability Test

- We use the following inequality to check if a task set is schedulable
  \[ u_1(1) + u_2(2) - u_1(1)[u_2(2) - u_2(1)] \leq 1 \]
  - \( x \) must be chosen in the following range
    \[ x \in \left[ \frac{u_2(1)}{1 - u_1(1)}, \frac{1 - u_2(2)}{u_1(1)} \right] \]

\( u_1(1) = \frac{1}{2} \)
\( u_2(1) = \frac{1}{8} + \frac{1}{9} \)
\( u_2(2) = \frac{3}{8} + \frac{3}{9} \)
EDF-VD for Arbitrary-Deadline Tasks

- Extend to arbitrary-deadline sporadic tasks
  - The task system is schedulable if
    - $L_1 + L_2 / 2 \leq 1$
    - $[L_1 L_2 - 2L_1]^2 - 4L_1[L_1 - L_2 + L_2^2] \geq 0$
    - $x$ must be $1 - L_2 / 2$
  - It’s not implicit-deadline any more. Thus load is used to check the schedulability
EDF-VD Algorithm Speedup Factors

- The speedup factor of EDF-VD on implicit-deadline tasks with two levels is **1.33**
  - This is also shown to be optimal with respect to speedup factors

- The speedup factor of EDF-VD on arbitrary-deadline tasks with two levels is **1.866**
  - It’s proved that no better bound can be found
    - Optimal speedup factor is 1.618
    - EDF-VD is not optimal for arbitrary-deadline tasks
Summary of Dissertation Research

- **OCBP algorithm for one-shot jobs**
  - Applicable to arbitrary number of criticality levels
  - Optimal with respect to speedup factors for two criticality levels

- **EDF-VD algorithm for sporadic tasks in two criticality levels**
  - Applicable to implicit and arbitrary deadlines
  - Optimal with respect to speedup factors for implicit-deadline tasks
Outline

- Motivation
- Thesis statement
- Background
- Dissertation research
  - Models
  - Algorithms
- Other contributions and future work
Other Contributions

- **OCBP algorithm on sporadic tasks [RTSS 10]**
  - OCBP algorithm is extended to sporadic tasks
    - The speedup factor results are maintained
    - My solution has a pseudo-polynomial run-time complexity. It is then improved to $O(n^2)$ [Guan et al., RTSS 11]


Other Contributions

- **EDF-VD algorithm on multiprocessors**
  - EDF-VD algorithm is extended to multiprocessor platforms
    - **Partitioned** multiprocessor scheduling [RTS 12]
    - **Global** multiprocessor scheduling [ECRTS 12]


Other Contributions

- **EDF-VD algorithm on multiprocessors**
  - **Partitioned multiprocessor scheduling** [RTS 12]
    - Each task is allocated to a processor
    - Jobs (and tasks) can not migrate from a processor to another
  - **Global multiprocessor scheduling** [ECRTS 12]
    - Jobs can migrate to and execute on different processors
  - **Both algorithms are for implicit-deadline tasks**
Other Contributions

- **EDF-VD algorithm on multiprocessors**
  - **Partitioned multiprocessor scheduling** [RTS 12]
    - The speedup factor is shown to be no greater than 2.67
  - **Global multiprocessor scheduling** [ECRTS 12]
    - The speedup factor is shown to be no greater than 3.24
  - The traditional partitioned and global EDF method has a speedup factor of 4
Future Work

- More efficient uniprocessor algorithms
  - Pragmatic improvement to OCBP algorithm
  - More criticality levels for EDF-VD algorithm

- Exploration of multiprocessor algorithms
  - Arbitrary-deadline tasks
  - More criticality levels
  - Better analysis and better algorithms
Thank you!