# Scheduling Mixed-Criticality Real-Time Tasks with Fault Tolerance

Jian (Denny) Lin[1], Albert M. K. Cheng[2,*], Douglas Steel[1], Michael Yu-Chi Wu[1]

[1] University of Houston – Clear Lake, USA

[2] University of Houston, USA

[*]Supported by the National Science Foundation under Awards No. 0720856 and No. 1219082.

## Outline

- Introduction – What is a mixed-criticality real-time system? Plus previous work.
- A static algorithm to increase the number of schedulable tasks in a mixed-criticality, fault-tolerant real-time system.
- A dynamic, on-line algorithm to reduce the likelihood of missing task deadlines.
- Experimental results.

# Mixed-Criticality, Real-Time Systems

- Integration of multiple functionalities on a single hardware platform
- Tasks running on these systems share resources but they have different importance (criticality)
- Tasks may require different levels of assurance, which may result in different, estimated Worst Case Execution Time (WCET)
  - WCET estimated by the Certification Authorities (CA)
  - WCET estimated by system designers
- Conventional scheduling methods cannot satisfactorily address the issues of scheduling mixed-criticality, real-time tasks

## A Classical Real-Time Task System

- Each task is periodic and characterized by a 4-tuple:

$$T_i = (p_i; X_i; c_i(LO); c_i(HI))$$

  - $p_i$ is the period of $T_i$
  - $X_i$ denotes the criticality of $T_{i,}$ either HI or LO
  - A HI-criticality task may have two different WCETs where $c_i(HI) >= c_i(LO)$, for example, two different estimated WCETs by CA and system designer.
  - A LO-criticality task has only one WCET $c_i(LO)$

## A Classical Real-Time Task System Cont.

- After a system starts running, every (periodic) task has an infinite sequence of jobs to execute.
- Initially, all HI-criticality and LO-criticality tasks in the system are scheduled using their c(LO)s and in this stage the system is said to be in the LO-criticality mode.
- A HI-criticality task may signal that its execution time exceeds its c(LO). At this point, all HI-criticality tasks will assume their c(HI)s and the system will go into the HI-criticality mode.
- In the HI-criticality mode, all LO-criticality tasks are dropped in order to maintain the safety of running the HI-criticality tasks.
    - Does it have to drop all the LO-criticality tasks? Probably not.

# EDF-VD (virtual deadline) Algorithm

- An algorithm using EDF to schedule mixed-criticality tasks
- Two different deadlines are used for some tasks if they exhibit two different WCETs during run-time. A shorter deadline, called virtual deadline, is used in the LO-criticality mode while the original deadline is used in the HI-criticality mode.
  - to ensure that the system is schedulable during mode switching:

$$U_x^y = \sum_{X=x} \frac{C(y)}{p}$$

- A notation for Utilization:
  - The superscript denotes the mode type and the subscript denotes the task type:
  - For example: $U_{HI}^{LO}$ denotes the total utilization of the HI-criticality tasks based on their C(LO)s.

## EDF-VD (virtual deadline) Algorithm Cont.

- How to calculate the virtual deadline?
  - A virtual deadline used for each HI-criticality task in the LO-criticality mode is obtained off-line with a scaling factor x, where the virtual relative deadline of the HI-criticality is equal to x * p (x times by the period).
  - x is defined as: $$x \in [\frac{U_{HI}^{LO}}{1 - U_{LO}^{LO}}, \frac{1 - U_{HI}^{HI}}{U_{LO}^{LO}}]$$

- Reference: S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster and L. Stougie, "The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Tas Systems", in ECRTS, pages 147-155, 2008.

## Fault-Tolerance

- Computation quality is also clearly important for a real-time system with critical tasks.
- Faults or errors may happen, leading to incorrect results or causing critical tasks to miss deadlines.
- Transient faults can be tolerated by adding redundancy where a task will be re-executed if it completes with errors.
  - Primary execution and re-execution model.

University of Houston Clear Lake

## Problems Studied in This Work

- Schedule a set of real-time, fault-tolerant tasks in a mixed-criticality system using EDF.
  - HI-criticality tasks are scheduled for their primary executions and re-executions, in both LO-criticality and HI-criticality mode.
  - How about LO-criticality tasks? LO-criticality tasks are not useless. We may not have to drop all of them in the HI-criticality mode as in the classical task model and EDF-VD algorithm.
  - How to exploit slack during the run-time?

University of Houston Clear Lake

## A Static Algorithm to Maximize Scheduled Tasks

- Notation used in our work:

$$U_x^y(a) = \sum_{a \in \{pri, re\} \wedge X_i = x} \frac{c_i(y)}{p_i}$$

- For a non-trivial problem, all tasks are schedulable in the LO-criticality mode with the following condition:

$$U_{HI}^{LO}(pri) + U_{HI}^{LO}(re) + U_{LO}^{LO}(pri) + U_{LO}^{LO}(re) \leq 1$$

- Applying the concept of the EDF-VD, all HI-criticality tasks are scheduled as long as x is between the following boundaries :

$$x \geq \frac{U_{HI}^{LO}(pri) + U_{HI}^{LO}(re)}{1 - U_{LO}^{LO}(pri) - U_{LO}^{LO}(re)}$$

$$x \leq \frac{1 - (U_{HI}^{HI}(pri) + U_{HI}^{HI}(re))}{U_{LO}^{LO}(pri) + U_{LO}^{LO}(re)}$$

## A Static Algorithm to Maximize Scheduled Tasks Cont.

- The ideas behind our algorithm
  - Any value between the two boundaries can be used for x.
  - There is room between these two boundaries so that it can be used to keep some LO-criticality tasks' executions in the system's HI-criticality mode.
  - A scheduled LO-criticality task can be taken as a special type of HI-criticality task because its deadline is guaranteed to be met.
  - The problem is similar to the bin-packing problem. When a LO-criticality task is scheduled with the HI-criticality tasks, the gap between the two boundaries is narrowed. A smallest-first strategy can be used until no more LO-criticality tasks can be added to the execution with the HI-criticality tasks.

## A Static Algorithm to Maximize Scheduled Tasks Cont.

* When statically scheduling LO-criticality tasks, we try to schedule their primary executions before their re-executions, assuming the error rate is not high.
* (pri)' means the primary executions of LO-criticality tasks that are reserved to schedule.
* (pri)'' means the primary execution that are not reserved (e.g., the LO-criticality tasks that may be discarded during the HI-criticality mode.

$$x \geq \frac{U_{HI}^{LO}(pri) + U_{HI}^{LO}(re) + U_{LO}^{LO}(pri)'}{1 - U_{LO}^{LO}(pri)'' - U_{LO}^{LO}(re)}$$

$$x \leq \frac{1 - (U_{HI}^{HI}(pri) + U_{HI}^{HI}(re) + U_{LO}^{LO}(pri)')}{U_{LO}^{LO}(pri)'' + U_{LO}^{LO}(re)}$$

## A Static Algorithm to Maximize Scheduled Tasks Cont.

- If the resource sufficiently allows all LO-criticality tasks to have their primary executions reserved, we can try to reserve more re-executions for LO-criticality tasks.

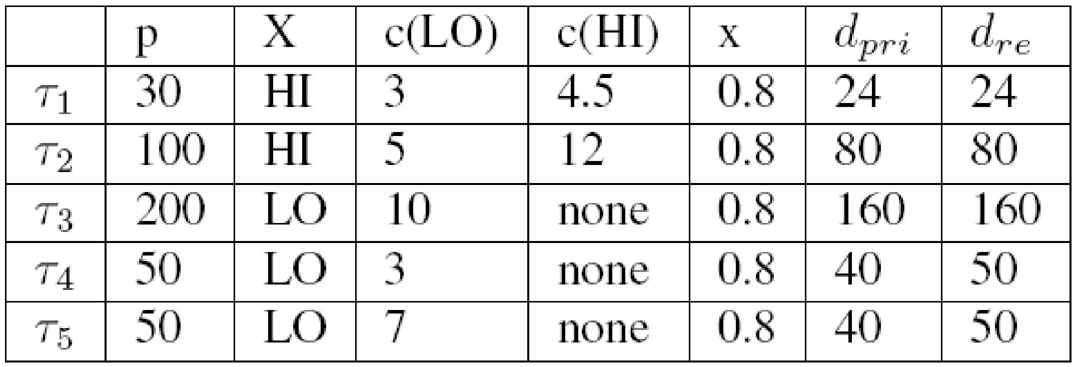
$$x \geq \frac{U_{HI}^{LO}(pri) + U_{HI}^{LO}(re) + U_{LO}^{LO}(pri) + U_{LO}^{LO}(re)'}{1 - U_{LO}^{LO}(re)''}$$

$$x \leq \frac{1 - (U_{HI}^{HI}(pri) + U_{HI}^{HI}(re) + U_{LO}^{LO}(pri) + U_{LO}^{LO}(re)')}{U_{LO}^{LO}(re)''}$$

- Our static algorithm tries to schedule additional LO-criticality tasks one-by-one. Each time, the two boundaries are calculated again. The number of schedulable tasks is maximized while the difference between these two boundaries is nearest to zero.
- While the executions of LO-criticality tasks (pri and/or re-execution) are scheduled, they work as HI-criticality tasks which have two different deadlines used at run-time.

## An Example Using Our Static Algorithm

- Instead of discarding all LO-criticality tasks, all LO-criticality primary tasks and $T_3$'s re-execution task are schedulable. The last two columns show the calculated virtual deadlines.



| | p | X | c(LO) | c(HI) | x | $d_{pri}$ | $d_{re}$ |
|---|---|---|---|---|---|---|---|
| $\tau_1$ | 30 | HI | 3 | 4.5 | 0.8 | 24 | 24 |
| $\tau_2$ | 100 | HI | 5 | 12 | 0.8 | 80 | 80 |
| $\tau_3$ | 200 | LO | 10 | none | 0.8 | 160 | 160 |
| $\tau_4$ | 50 | LO | 3 | none | 0.8 | 40 | 50 |
| $\tau_5$ | 50 | LO | 7 | none | 0.8 | 40 | 50 |

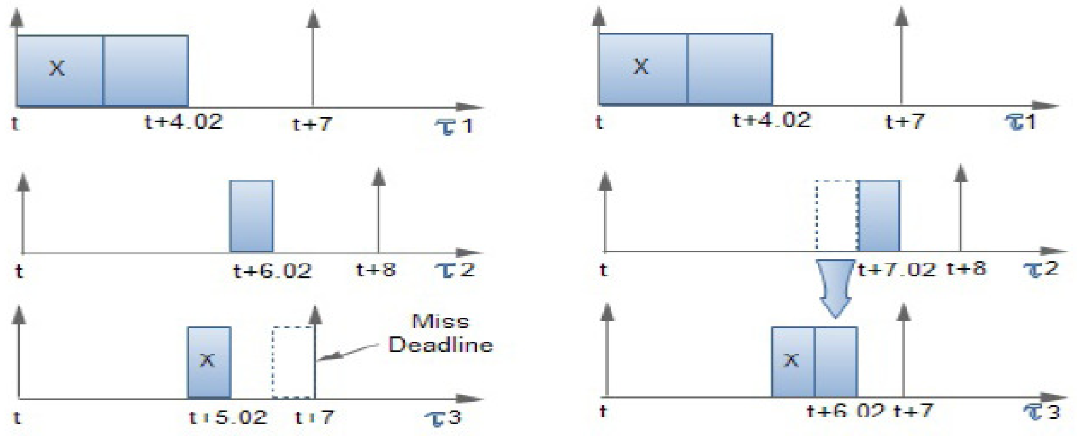
Table 1. Calculated relative deadlines of primary and re-executions of a 5-tasks set

## An On-Line Slack Reclaiming Algorithm

- The strategy of using re-executions to preserve the reliability is relatively conservative. If there are no errors, the resource for the reserved re-executions is wasted. Also, in most cases, a real-time task does not use up its WCET.
  - The slack generated can be used to execute more LO-criticality task jobs.
- When a fault is detected by a LO-criticality job without a re-execution reserved, it may also use the slack from future jobs.

# A Motivational Example for Using Future Slack

- Three tasks in the HI-criticality mode: T1 is a HI-criticality task with c1(HI) = 2.01, and T2 and T3 are two LO-criticality tasks with the same c(LO) = 1. Both 1 and 2 have a re-execution reserved. The periods of these three tasks are 7, 8, and 7.
- Suppose an error occurs in T3 as shown in the figure. A re-execution of T3 is executed by assuming that no error occurs in T2's job.



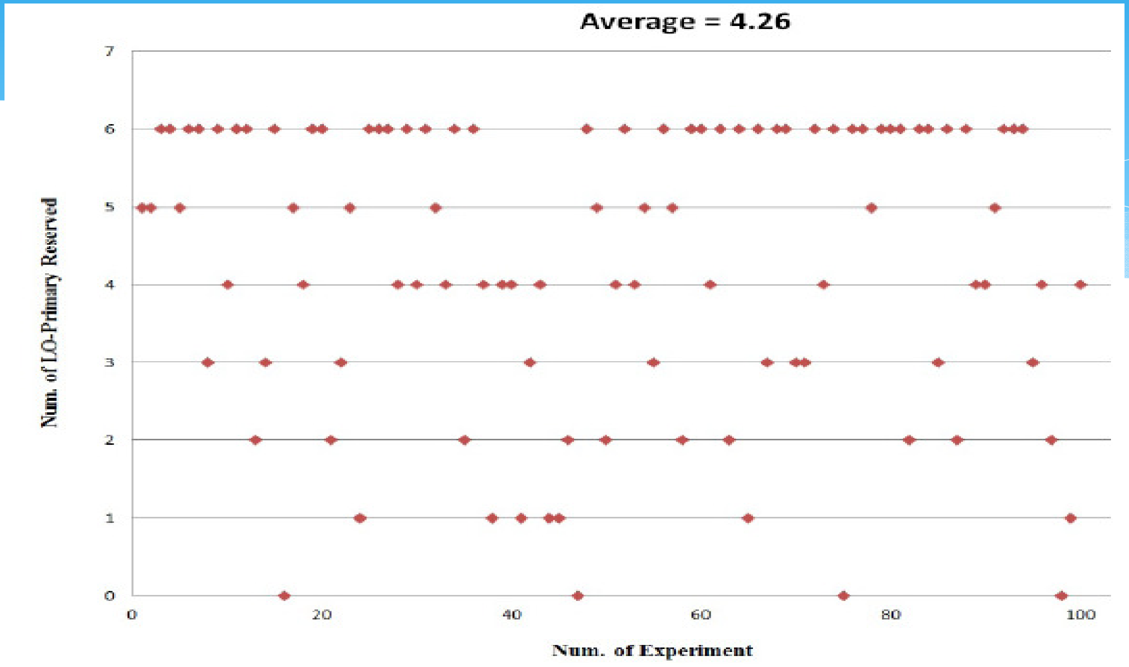
## A Motivational Example for Using Future Slack Cont.

- Even if the job of T2 ends with an incorrect result due to a lack of re-execution, the consequence is not catastrophic because both jobs are not HI-criticality.
- Considering the small likelihood of a fault actually occurring, the idea behind this solution has the potential to increase the overall reliability and system's performance.

## Safely Use Future Slack

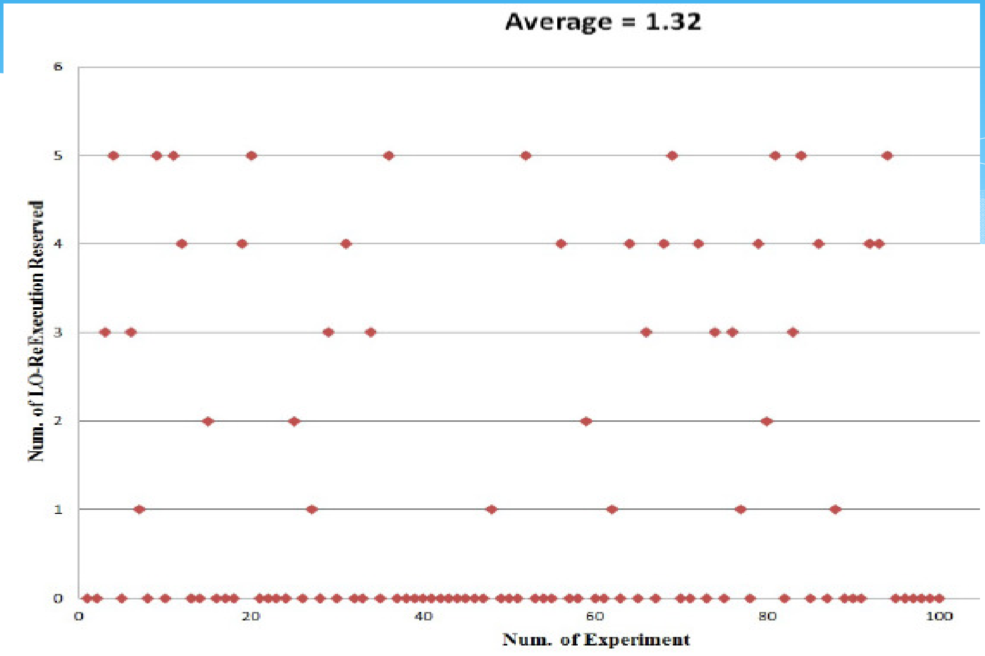
- Use slack generated from early completion (including no operation of re-execution because of completing without errors) first. This type of slack is the safest to use.
- Only use LO-criticality tasks' future slack. HI-criticality should not be risked.
- A server-based scheduler is used to manage the slack generated from early completions.

# Experimental Results



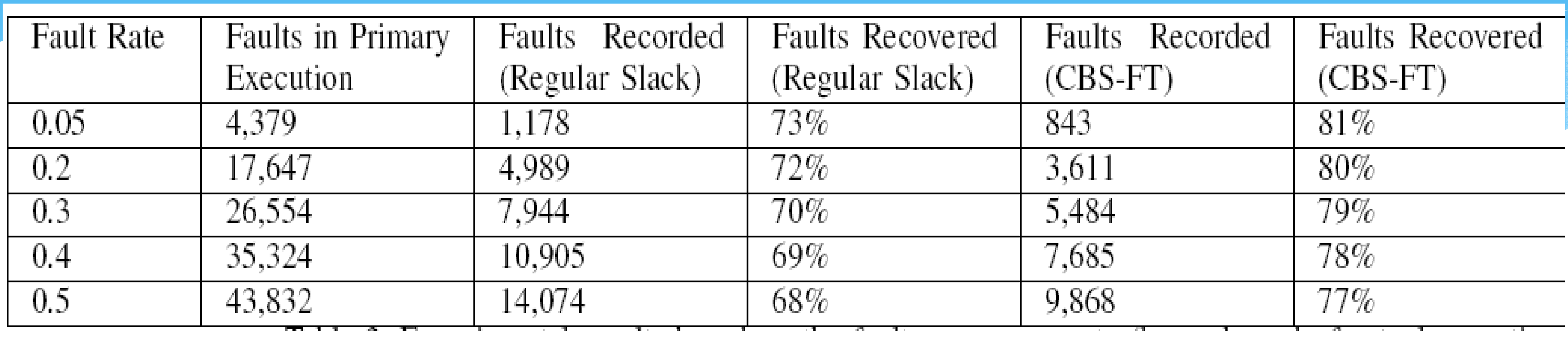
- 100 task sets per experiment
- Each task set has 10 randomly generated tasks, 4 HI-criticality tasks and 6 LO-criticality tasks
- For each task set, we use our static algorithm to calculate how many LO-criticality primary tasks can be scheduled
- The x-axis is the index of the task set and the y-axis is the number of LO-criticality primary tasks scheduled

## Experimental Results Cont.



- We also calculated how many LO-criticality re-executions can be scheduled if all LO-criticality primary tasks are scheduled

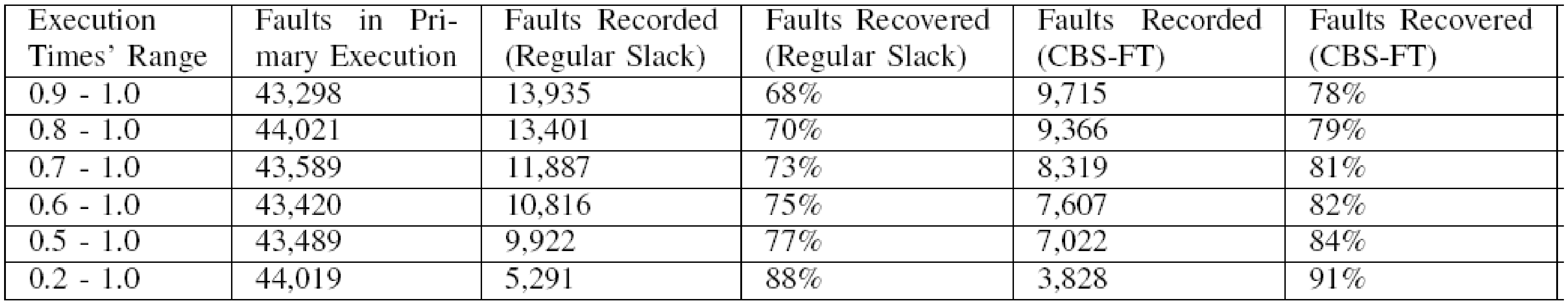
# Experimental Results Cont.



| Fault Rate | Faults in Primary Execution | Faults Recorded (Regular Slack) | Faults Recovered (Regular Slack) | Faults Recorded (CBS-FT) | Faults Recovered (CBS-FT) |
|---|---|---|---|---|---|
| 0.05 | 4,379 | 1,178 | 73% | 843 | 81% |
| 0.2 | 17,647 | 4,989 | 72% | 3,611 | 80% |
| 0.3 | 26,554 | 7,944 | 70% | 5,484 | 79% |
| 0.4 | 35,324 | 10,905 | 69% | 7,685 | 78% |
| 0.5 | 43,832 | 14,074 | 68% | 9,868 | 77% |

- Fault Rate: how likely an error occurs in execution
- CBS-FT (Constant Bandwidth Server – Fault Tolerance): our online algorithm using future slack for fault tolerance
- The numbers of faults recovered by using early completion slack only and using future slack also are compared

## Experimental Results Cont.



| Execution Times' Range | Faults in Primary Execution | Faults Recorded (Regular Slack) | Faults Recovered (Regular Slack) | Faults Recorded (CBS-FT) | Faults Recovered (CBS-FT) |
|---|---|---|---|---|---|
| 0.9 - 1.0 | 43,298 | 13,935 | 68% | 9,715 | 78% |
| 0.8 - 1.0 | 44,021 | 13,401 | 70% | 9,366 | 79% |
| 0.7 - 1.0 | 43,589 | 11,887 | 73% | 8,319 | 81% |
| 0.6 - 1.0 | 43,420 | 10,816 | 75% | 7,607 | 82% |
| 0.5 - 1.0 | 43,489 | 9,922 | 77% | 7,022 | 84% |
| 0.2 - 1.0 | 44,019 | 5,291 | 88% | 3,828 | 91% |

* Execution Times' Range: how varied the real execution time compared with its WCET
* The numbers of faults recovered by using early completion slack only and using future slack also are compared

# Real-Time Systems Group

- **Director** Prof. Albert M. K. Cheng
- **PhD students**
  Yong Woon Ahn, Yu Li, Xingliang Zou, Behnaz Sanati, Sergio Chacon, Zeinab Kazemi, Carlos Rincon, Qiong Lu, Seyed Mohsen Alavi (arriving in spring 2015)
- **MS students**
  Daxiao Liu, Chonghua Li
- **Undergraduate students (NSF-REU)**
  Mozahid Haque, Rachel Madrigal
- **Visiting scholars**
  Yu Jiang (Heilongjiang U.), Qiang Zhou (Beihang U.), Yufeng Zhao (Xi'an Tech. U.)
- **Recent graduates and their positions**
  Yuanfeng Wen (MS, Microsoft), Chaitanya Belwal (PhD, Visiting Assistant Professor, UHCL), Jim Ras (PhD), Jian Lin (PhD, Assistant Professor, UHCL )

Yu Li (Best Junior PhD Student Awardee and Friends of NSM Graduate Fellow) and Prof. Albert Cheng visit the NSF-sponsored Arecibo Observatory after their presentation at the flagship RTSS 2012 in Puerto Rico.

Real-time systems research group at Yuanfeng Wen's graduation party in May 2013. Yuanfeng is now at Microsoft.

Fall 2014 (9/3) group meeting - from left to right: Dr. Qiang Zhou, Qiong Lu, Carlos Rincon, Chonghua Li, Prof. Yu Jiang, Xin Liu, Prof. Yufeng Zhao, Prof. Albert Cheng, Xingliang (Jeffrey) Zou, Daxiao Liu, Yu Li, Yong Woon Ahn, and Behnaz Sanati. Zeinab Kazemi in class.