LITMUS NOTES

Paper clippings

May 18, 2017

Contents

1 EDF-VD

1.1 Overview

1.2 Offline Component

1.3 Online Component

1.4 Litmus Changes

2 Elastic MC

2.1 Overview

2.2 Offline Component

2.3 Online Component

2.4 Litmus Changes

3 Adaptive Mixed Criticality(AMC) Scheduling.

3.1 Overview

3.2 Offline Component

3.3 Online Component

3.4 Litmus Changes

4 AMC with Deferred Preemption

4.1 Overview

This paper builds upon Adaptive Mixed Criticality(AMC) scheduling. A final non preemptive region is introduced to improve the schedulability of a system with multiple criticality levels.

4.2 Offline Component

4.3 Online Component

4.4 Litmus Changes

5 Bailout Protocol

5.1 Overview

5.2 Offline Component

5.3 Online Component

5.4 Litmus Changes

6 Interference Constraint Graph.

6.1 Overview

Classical Mixed Criticality algorithms provide assurances to meet budget requirement of high critical tasks at the cost of dropping low criticality tasks in case of system criticality change. Other scheduling approaches have proposed means of improving guarantees to low critical tasks. But these approaches does not take into consideration the which tasks interfere with each other and in which order, which is very much application specific.
This scheduling algorithm enables to take task interference specifications from system designed and create a schedule that meets the specific requirement.

6.2 Offline Component

The offline component of the proposed algorithm are mainly three:

- A task model to represent the dependency between the tasks as graph.
- A demand bound function(DBF) that is compatible with Audsley's priority assignment approach.
- An algorithm to minimize the interference between the tasks.

6.3 Online Component

The online component of the algorithm builds upon standard fixed priority scheduling.

- Task structure is extended to provide support for variable representing budgets and deadline for a multiple criticality taskset. Though ICG presents the approach for arbitrary number of criticality, Dual criticality is considered for online implementation.
- Each task maintains a list of pointers to tasks that can be dropped in case of a budget overrun.
- For a budget overrun, the list is walked through and task_struct variable *skipped* flag is set to 1. These tasks are simply requeued when picked from run queue.
-

6.4 Litmus Changes

liblitmus change

- Each task instance of rtspin has to be notified of the tasks that can be penalized in case of budget overruns. This is done in two stages.

- Each new rtspin is assigned an incrementing ID(integer index value) to identify itself and a set of indexes corresponding to the peer rtspin tasks that it can interfere with during an overrun scenario. These IDs are passed to the litmus scheduler as part of *task_struct* variable.

litmus-rt scheduler plugin

- Within litmus scheduler the IDs and dependent IDs are used to create a task dependency list for each *task_struct* which consists of all the *task_struct* pointers of dependent tasks.
- During an active schedule, if one of the tasks overruns its budget, then all the *task_struct* in dependency list are updated. *skip* parameter is set to 1.
- For each invocation of *schedule* function in the ICG Plugin, *skip* parameter is checked. While the parameter is set to 1, the task is not scheduled instead rescheduled for next release instance.
- Parameter knobs are provided to tweak the behavior in case of budget overrun. As per the inference of the example 4.1 in section 4.1, the skipping of the task can be enabled after a minimum number of budget overruns(e.g. k overruns in n consecutive instances.)

7 Preemption Threshold.

7.1 Overview

7.2 Offline Component

7.3 Online Component

7.4 Litmus Changes

8 Service Adaptation.

8.1 Overview

For an task execution scenario consisting of tasks of different criticalities, Mixed Criticality algorithms strives to guarantee worst case execution requirements. One common approach towards this guarantee is dropping tasks of lower criticality when a higher criticality task is not able to meet it's WCET. This approach is overly pessimistic, penalizing lower criticality tasks. Another approach to this problem has been to do best effort scheduling for lower criticality tasks in case of a budget overrun. But a real life deployment of such tasks consisting of a mixture of low and high criticality calls for guaranteeing a minimum service even for low criticality tasks. This paper:

- Extends EDF-VD scheduling technique to guarantee degraded service for low criticality tasks in high mode.
- Extend demand bound analysis of MC systems, scheduled by mode switch EDF.
- Offline bound to determine resetting time for services provided to low criticality tasks.

8.2 Offline Component

The offline component encompasses 3 main contributions:

8.2.1 Demand Bound Function

A generic representation of Demand Bound Function(DBF) is proposed, which takes into account all the tasks that are present in high criticality.
Set of equation to calculate the DBF are given below:

$$\mathbf{dbf_{LO}} = max\{\lfloor\frac{\Delta - D_i(LO)}{T_i(LO)}\rfloor + 1, 0\}.C_i(LO) \tag{1}$$

$$\boldsymbol{RM}(\tau_i, \lambda) = C_i(HI) - C_i(LO) + minD_i(LO) - \lambda, C_i(LO) \tag{2}$$

$$\mathbf{dbf}^1{}_{HI}(\tau_i, \Delta) = max\{\lfloor\frac{\Delta - D_i(HI) + 1, 0}{T_i(HI)}\rfloor\}.C_i(HI) \tag{3}$$

$$\mathbf{dbf}_{RM}(\tau_i, \lambda, \Delta) = \begin{cases} RM(\tau_i, \lambda) & if \quad \Delta \geq D_i(HI) - \lambda, \\ 0 & if \quad \Delta < D_i(HI) - \lambda \end{cases} \tag{4}$$

$$\mathbf{dbf}^2{}_{HI}(\tau_i, \lambda, \Delta) = dbf_{RM}(\tau_i, \lambda, \Delta) + max\{\lfloor \frac{\Delta - D_i(HI) - (T_i(HI) - \lambda)}{T_i(HI)} \rfloor + 1, 0\}.C_i(HI) \quad (5)$$

$$\mathbf{dbf}_{HI}(\tau_i, \Delta) = max\{dbf^1{}_{HI}(\tau_i, \Delta), \quad max_{0 \leq \lambda \leq D_i{}_{LO}}\{dbf^2{}_{HI}\}\} \quad (6)$$

Equation 8 gives the DBF of any task in HI mode.

8.2.2 Service Reconfiguration.

Low criticality tasks are guaranteed a minimum service in high criticality mode by using a scaling factor. For this purpose two new variables are introduced: service degradation factor y for low criticality tasks and deadline tuning factor x.

8.3 Online Component

The online component is based on EDF-VD scheduling approach, where low critical tasks are dropped, when a budget overrun in a high critical task is detected. This paper proposes two approach to recover system from a high critical mode:

- Recovery at idle instance.
- Recovery based on MC Service reset time.

8.4 Litmus Changes

The core kernel scheduler will use EDF-VD implementation, following further changes are introduced for supporting service adaptation:

- Schedcat implementation of the demand bound function and schedulability test.
- Schedcat support for calculating the deadline tuning factor x and service degradation factor y.
- rtspin changes to set the value of MC Service reset time.
- litmus-rt edfvd scheduler extended to support MC service reset time.

9 Study

9.1 Design and analysis for Dual Priority Scheduling

9.1.1 Idea

Hard Real-Time systems often employ priority driven preemptive scheduling for efficient usage of system resource. And in this context preemptive dynamic priority based Earliest Deadline

First(EDF is known to be optimal for scheduling such tasks. But approach in industry has been towards using Fixed priority scheduling due to its efficient runtime support. Multiple approaches have been considered to improve the schedulability under FP scheduling and one such approach is priority promotion based scheduling.

9.1.2 Offline Component

9.1.3 Online Component

9.1.4 Litmus Changes

9.1.5 Reference

`http://codeventure.sce.ntu.edu.sg/teaching/2017/ce7452/projects/project_papers/dual_priorit.pdf`

9.2 Resource Efficient Isolation Mechanisms in Mixed-Criticality Scheduling

9.2.1 Idea

9.2.2 Offline Component

9.2.3 Online Component

9.2.4 Litmus Changes

9.2.5 Reference

`http://ieeexplore.ieee.org/document/7176022/`

10 Implementation

11 Experiments

12 Additional References

12.1 Design Optimization of Mixed-Criticality Real-Time Embedded Systems

12.2 Design of an Efficient Ready Queue for Earliest-Deadline-First (EDF) Scheduler

12.2.1 Main Idea

A new ready queue design to improve the performance of the edf scheduler.

13 Experiments and Design

References