Efficiently Safe: Decoding the Dichotomy in Mixed-Criticality Systems

Arvind Easwaran

Nanyang Technological University, Singapore

April 15, 2015
Outline

Mixed-Criticality System (MCS)

Role of Components in MCS

Task Model with Multiple Budgets

Research Trends & Challenges
What is a Mixed-Criticality System (MCS)?

A system with multiple applications that are “certified” to different levels of criticality and share hardware resources.
What is a Mixed-Criticality System (MCS)?

A system with multiple applications that are “certified” to different levels of criticality and share hardware resources

- An automotive example → Adaptive Cruise Control (ACC), a highly critical application, and Battery Management, a less critical application, sharing hardware
- MCS has been around for a while (e.g., Integrated Modular Avionics was introduced commercially in the 1990s)
What is a Mixed-Criticality System (MCS)?

A system with multiple applications that are “certified” to different levels of criticality and share hardware resources

- An automotive example → Adaptive Cruise Control (ACC), a highly critical application, and Battery Management, a less critical application, sharing hardware
- MCS has been around for a while (e.g., Integrated Modular Avionics was introduced commercially in the 1990s)

Important to distinguish between less critical and non-critical applications

- Infotainment is probably non-critical, and should therefore be treated differently
Partitioning for Isolation

Worst-case resource reservation has been used as an effective strategy for achieving isolation in MCS

- Allocates hardware resources to applications
- Uses runtime mechanism for enforcing the reservations (i.e., scheduler)
- Time- and space-partitioned systems
- Examples: IMA, AUTOSAR
Efficient Resource Usage

Resource reservations are generally very pessimistic (high criticality $\implies$ high pessimism)

- Pessimism ensures safety in the presence of complexity
- Leads to severe under-utilization of hardware resources
Efficient Resource Usage

Resource reservations are generally very pessimistic (high criticality → high pessimism)

- Pessimism ensures safety in the presence of complexity
- Leads to severe under-utilization of hardware resources

Key challenge for MCS is how to achieve resource efficiency without compromising on safety

- A safety-critical application will always receive resources
- A critical application that is not crucial for safety, may not
Component-based MCS

Many MCS are in fact developed using the component-based design paradigm

- Not surprisingly, because they are comprised of several applications
- Each application is independently developed and then integrated into the system
Component-based MCS

Many MCS are in fact developed using the component-based design paradigm

- Not surprisingly, because they are comprised of several applications
- Each application is independently developed and then integrated into the system

This design paradigm has several benefits

- Fault containment to ensure safety (similar to resource reservation, but in the functional domain)
- Simplication of design (well defined interactions between components)
A Simplistic View of Components in MCS

**Component 1**
High-Criticality (HC) Application
(one or more HC real-time tasks)

**Component 2**
Low-Criticality (LC) Application
(one or more LC real-time tasks)

- Tasks are distributed over a networked hardware platform
- Tasks are certified collectively as a component
- Intra- as well as inter-criticality dependencies
- Prevalent view as of today
A Different View of Components in MCS

Component 1

High-Criticality (HC) Application
(one or more HC and LC real-time tasks)

Component 2

Low-Criticality (LC) Application
(one or more LC real-time tasks)

- Engine Controls with Condition Based Monitoring (CBM)
  - CBM could be certified to a lower level
  - Certifying parts of a HC component to a lower-criticality has a huge monetary benefit, if it can be done
  - Nevertheless, component boundaries are important
Worst-Case Execution Time (WCET) estimation is crucial for budgeting/reservation

- Hard to estimate accurately for MCS (complexity)
- Common approach is a combination of the following
  1. Conduct measurements (stress testing, cache thrashing)
  2. Perform analysis (profiling, path analysis)
Worst-Case Execution Time (WCET) estimation is crucial for budgeting/reservation

- Hard to estimate accurately for MCS (complexity)
- Common approach is a combination of the following
  1. Conduct measurements (stress testing, cache thrashing)
  2. Perform analysis (profiling, path analysis)

Use engineering judgement (fudge factor or padding) to compute task budgets based on estimated WCETs

- Higher the criticality, higher is the fudge factor for safety
Budget Determination for Critical Tasks

Worst-Case Execution Time (WCET) estimation is crucial for budgeting/reservation

- Hard to estimate accurately for MCS (complexity)
- Common approach is a combination of the following
  1. Conduct measurements (stress testing, cache thrashing)
  2. Perform analysis (profiling, path analysis)

Use engineering judgement (fudge factor or padding) to compute task budgets based on estimated WCETs

- Higher the criticality, higher is the fudge factor for safety
- Higher the criticality, higher is the pessimism in the budget
  $\implies$ lower is the resource efficiency
Vestal Task and Execution Model

1. Reserve less “pessimistic” budget for high-critical tasks
   - Do not use engineering judgement, or use less of it!
Vestal Task and Execution Model

1. Reserve less “pessimistic” budget for high-critical tasks
   - Do not use engineering judgement, or use less of it!

2. In the (hopefully) “rare” case that budget is exceeded, give additional budget to high-critical tasks
Vestal Task and Execution Model

1. Reserve less “pessimistic” budget for high-critical tasks
   - Do not use engineering judgement, or use less of it!

2. In the (hopefully) “rare” case that budget is exceeded, give additional budget to high-critical tasks

3. Continue to certify high-critical tasks as before assuming pessimistic budgets
Vestal Task and Execution Model

1. Reserve less “pessimistic” budget for high-critical tasks
   - Do not use engineering judgement, or use less of it!

2. In the (hopefully) “rare” case that budget is exceeded, give additional budget to high-critical tasks

3. Continue to certify high-critical tasks as before assuming pessimistic budgets

4. But certify low-critical tasks using more realistic budgets for all the tasks
   - When high-critical tasks require additional budget, then low-critical tasks will be impacted
Research Trends in MCS

- Initial studies focused on improving low-criticality schedulability at design time
  - Did not consider runtime implications on low-criticality jobs
Research Trends in MCS

- Initial studies focused on improving low-criticality schedulability at design time
  - Did not consider runtime implications on low-criticality jobs

- Since then, a majority of the studies assume that once any high-criticality task requires additional budget
  - All of them will simultaneously require additional budgets
  - Consequently, all the low-criticality tasks will be suspended
Research Trends in MCS

- Initial studies focused on improving low-criticality schedulability at design time
  - Did not consider runtime implications on low-criticality jobs

- Since then, a majority of the studies assume that once any high-criticality task requires additional budget
  - All of them will simultaneously require additional budgets
  - Consequently, all the low-criticality tasks will be suspended

- Some recent studies focus on reducing the impact to low-criticality tasks at runtime, e.g., elastic LC releases, explicit dependency specification between HC and LC, etc.
Initial studies focused on improving low-criticality schedulability at design time
   - Did not consider runtime implications on low-criticality jobs

Since then, a majority of the studies assume that once any high-criticality task requires additional budget
   - All of them will simultaneously require additional budgets
   - Consequently, all the low-criticality tasks will be suspended

Some recent studies focus on reducing the impact to low-criticality tasks at runtime, e.g., elastic LC releases, explicit dependency specification between HC and LC, etc.

Worst-case resource reservation to achieve partitioning is still the most prevalent strategy in industry
Research Challenges in MCS (1)

Studies that focus on reducing the impact on low-criticality tasks at runtime is needed

- Why is it reasonable to assume that all the high-criticality tasks will simultaneously require additional budgets?

- Why suspend all the low-criticality tasks even when a single high-criticality task requires additional budget?

- What is the consequence of abruptly suspending low-criticality tasks?
  - Relation to mode change, graceful degradation
Research Challenges in MCS (2)

Studies that use component boundaries to limit the impact on low-criticality tasks is needed

- Can we limit the impact to within components as much as possible?
  - What component-level mechanisms are needed to limit the impact? Do they need to be compositional as well?
  - How effective would this be in a real system?

- Does hierarchical scheduling still play a role in MCS?
  - Criticality aware scheduling already guarantees partitioning
  - Can we meet compositional design goals without hierarchical scheduling?
Thank you for listening!