Runtime Software Selection for Adaptive Automotive Systems

Chia-Ching Fu, Ben-Hau Chia, Chung-Wei Lin

National Taiwan University
January 2021
Motivating Example: Deraining Algorithm

Should we execute a deraining algorithm for a vehicle?

- YOLOv3 [1] as the object detector
- PASCAL-VOC [2] as the dataset
- Progressive ResNet (PRN) [3] as the deraining algorithm

Motivating Example: Intrusion Detection

- **Cooperative Adaptive Cruise Control (CACC)**
  - Different detection approaches have different **detection performances** under **different scenarios** with **different computation times**

- **Examples [4, 5]**
  - Physics-based detection (PHY)
  - Principal Component Analysis (PCA) based detection
  - Hidden Markov Model (HMM) based detection

- **Observations**
  - Detection performance: HMM > PCA > PHY
  - Computation time: HMM > PCA > PHY

Adaptive Automotive Systems

- Automotive systems become more connected and autonomous than ever
  - There are more and more software programs running on systems

- Adaptive automotive systems, supported by over-the-air (OTA) updates and plug-and-play systems, provide great values
  - Software programs are updated, activated, and deactivated before driving and during driving

- However, not all software programs should be executed at the same time
  - Whether a software program should be executed depends on the environmental conditions
  - The computational resource on an automotive system is very limited
Related Work

❑ Automotive Open System Architecture (AUTOSAR) Adaptive Platform
   - The foundation of AUTOSAR runtime for adaptive applications
   - The lifecycles of automotive applications with AUTOSAR Adaptive Platform [6]

❑ Software update framework
   - Uptane, updating software on Electronic Control Units (ECUs) and addressing security and customizability [7]
   - SecUp, updating software based on IEEE 802.11s [8]
   - Integration between original equipment manufacturers and mobile service providers [9]

---

Contributions

❑ **A preliminary step towards the broad realization of adaptive automotive systems**
  - Consider the upcoming environmental conditions of an automotive system
  - Target the corresponding software selection problem during runtime
  - Formulate the problem as a set cover problem with Quality-of-Service (QoS) and timing constraints
  - Propose an efficient approach to solve the problem during runtime

❑ **Other possible scenarios**
  - Different traffic conditions (e.g., congestion)
  - Different weather conditions (e.g., raining, snowing)
  - Different regulations (e.g., crossing the borders of countries)
Outline

- Introduction
- **Problem Formulation**
- Proposed Approach
- Extension
- Experimental Results
- Conclusion
System Model

- **Vehicle**
  - The vehicle is connected so that it can collect the information of upcoming environments from roadside units or other vehicles
  - The vehicle is not necessarily autonomous
    - However, the vehicle usually has more functional tasks if it is autonomous

- **Environment**
  - There are multiple upcoming environments for the vehicle
  - Each environment has a set of functional tasks to be completed

- **Functional task**
  - There are multiple functional tasks under each environment
  - Each functional task with its QoS requirement needs to be satisfied by a software program

- **Software Program**
  - The vehicle maintains a set of software programs
  - Each software program can satisfy one or multiple functional tasks
Generality of System Model

- The system model is general to consider multiple upcoming environments at the same time
  - The vehicle selects software programs for "all" of the upcoming environments

**Special cases**
- (#1) The vehicle decides to perform software selection for each environment separately
- (#2) Each software program can only satisfy one functional task

**Assumption**
- There is a central computational unit running all software programs
  - The timing model can be replaced by another one integrating multiple computational units and in-vehicle networks
Problem Formulation (1/2)

Given
- A set of functional tasks
- A set of software programs
- A set of upcoming environments
- The period and computation time of each software program
- The QoS requirement of a functional task under an upcoming environment
- The QoS that a software program can provide for a functional task under an upcoming environment

Decision variables
- The selection (yes or no) of each software program
- The priority of each software program
Problem Formulation (2/2)

- **Dependent variables**
  - The worst-case response time of each software program
    - It is 0 if the software program is not selected

- **Objective**
  - Minimize the summation of the worst-case response times of all software programs

- **QoS Constraints**
  - The QoS requirement of each functional task is satisfied by at least one selected software program

- **Timing constraints**
  - The worst-case response time of each software program is smaller than or equal to its period
Outline

- Introduction
- Problem Formulation
- Proposed Approach
- Extension
- Experimental Results
- Conclusion
Step 1: Preprocessing

- $\tau_i$: functional task
- $\rho_j$: software program

QoS requirement

<table>
<thead>
<tr>
<th>QoS requirement</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
<th>$\tau_1$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\rho_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\rho_3$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\rho_4$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

QoS that a software program can provide
Step 2: Set Covering

- $\tau_i$: functional task
- $\rho_j$: software program
- Select $\rho_1$, $\rho_3$, $\rho_4$

<table>
<thead>
<tr>
<th>QoS requirement</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
<th>$\tau_1$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>1</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.8</td>
<td>1</td>
<td>0.8</td>
</tr>
</tbody>
</table>

QoS that a software program can provide
Step 2: Set Covering

- $\tau_i$: functional task
- $\rho_j$: software program
- Select $\rho_1$ and $\rho_2$

<table>
<thead>
<tr>
<th>QoS requirement</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\rho_1$</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\rho_2$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>$\rho_3$</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$\rho_4$</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

QoS that a software program can provide
### Step 2: Set Covering

- **A greedy approach selecting a software program which maximizes** \( \frac{N_j \cdot P_j}{C_j} \)
  - \( N_j \): number of "not-yet-satisfied" functional tasks that \( \rho_j \) can satisfy
  - \( C_j \): computational time of \( \rho_j \)
  - \( P_j \): period of \( \rho_j \)

<table>
<thead>
<tr>
<th>QoS requirement</th>
<th>( \tau_1 )</th>
<th>( \tau_2 )</th>
<th>( \tau_3 )</th>
<th>( \tau_1 )</th>
<th>( \tau_2 )</th>
<th>( \tau_3 )</th>
<th>( \tau_4 )</th>
<th>( \tau_1 )</th>
<th>( \tau_3 )</th>
<th>( \tau_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_1 )</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>( \rho_2 )</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \rho_3 )</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( \rho_4 )</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

QoS that a software program can provide
Steps 3 and 4

❑ **Step 3: Priority Assignment**
  ➢ Rate-monotonic scheduling
    • A software program with a smaller period has a higher priority

❑ **Step 4: Timing Analysis**
  ➢ Fixed-priority scheduling
    • \( r_j = C_j + \sum_{\text{higher-priority software program } j'} \left[ \frac{r_{j'}}{P_{j'}} \right] C_{j'} \)
    • \( r_j \): worst-case response time of \( \rho_j \)
    • \( C_j \): computational time of \( \rho_j \)
    • \( P_j \): period of \( \rho_j \)

❑ If the selected software programs are not schedulable, then our approach does not find a feasible solution
  ➢ In this case, the vehicle should switch to a conservative mode
Outline

- Introduction
- Problem Formulation
- Proposed Approach
- Extension
- Experimental Results
- Conclusion
Extension 1: Compatibility and Dependency

- **Original decision variables**
  - $x_j$: whether $\rho_j$ is selected
    - 1: selected
    - 0: not selected

- **Compatibility constraints**
  - If we select $\rho_j$, then we must "not" select $\rho_{j'}$.
    - SAT form: $\neg x_j \lor \neg x_{j'}$
  - It can be easily generalized to multiple software programs

- **Dependency constraints**
  - If we select $\rho_j$, then we must select $\rho_{j'}$.
    - SAT form: $\neg x_j \lor x_{j'}$
  - It can be easily generalized to multiple software programs
Extension 2: "Change" Minimization

- **System model**
  - Change selected software programs under each upcoming environment

- **Decision variables**
  - The selection (yes or no) of each software program under each upcoming environment
  - The priority of each software program under each upcoming environment

- **Objective**
  - Minimize the total number of changes
Outline

- Introduction
- Problem Formulation
- Proposed Approach
- Extension
- Experimental Results
- Conclusion
Test Cases

- 4 types of test cases and 10 test cases for each type
- The period and the computation time are selected from the given sets by uniform distributions
- A QoS requirement is assigned randomly and uniformly
- A provided QoS is assigned randomly with a positive correlation with the computation time

<table>
<thead>
<tr>
<th>Type</th>
<th># of Functional Tasks</th>
<th># of Software Programs</th>
<th># of Upcoming Environments</th>
<th>Period (ms)</th>
<th>Computation Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>32</td>
<td>8</td>
<td>{50, 100, 200}</td>
<td>{10, 20, 40}</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>32</td>
<td>8</td>
<td>{100, 200, 400}</td>
<td>{20, 40, 80}</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
<td>64</td>
<td>8</td>
<td>{50, 100, 200}</td>
<td>{10, 20, 40}</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>64</td>
<td>8</td>
<td>{100, 200, 400}</td>
<td>{20, 40, 80}</td>
</tr>
</tbody>
</table>
Comparative Approaches

- **Greedy**
  - Select a software program which maximizes $N_j$ (not $N_j \cdot P_j / C_j$)
    - $N_j$: number of "not-yet-satisfied" functional tasks that $\rho_j$ can satisfy
    - $P_j$: period of $\rho_j$
    - $C_j$: computational time of $\rho_j$

- **Greedy+Timing Analysis (TA)**
  - Perform the timing analysis after a software program is selected
  - If a timing constraint is violated, cancel the selection, and perform the greedy approach again

- **Computational platform**
  - A Windows Home 10 64-bit laptop with 1.8GHz Intel Core i7-8565U CPU and 8GB memory
The proposed approach achieves 100% successful rates

- It considers timing during software program selection and tends to use less computational resource to satisfy more functional tasks
- The other approaches have smaller objectives (if they find feasible solutions) by trading off the successful rates

Their execution times are small

- Even if the computational platform is downgraded, they are applicable to runtime computation and decision
Outline

- Introduction
- Problem Formulation
- Proposed Approach
- Extension
- Experimental Results
- Conclusion
Conclusion

- A preliminary step towards the broad realization of adaptive automotive systems
  - Considered the upcoming environmental conditions of an automotive system
  - Targeted the corresponding software selection problem during runtime
  - Formulated the problem as a set cover problem with QoS and timing constraints
  - Proposed an efficient approach to solve the problem during runtime

- Future work
  - The approaches for the two extended problems
  - Benchmarking of test cases
  - More complicated hardware platforms such as multi-core processors or multiple ECUs connected by in-vehicle networks
Thank You!

cwlin@csie.ntu.edu.tw