SAFER: System-level Architecture for Failure Evasion in Real-time Applications

Junsung Kim, Gaurav Bhatia and Raj Rajkumar
Carnegie Mellon University

Markus Jochim
General Motors
The Award-Winning Autonomous Vehicle

- Chevy Tahoe ("Boss")
  - Senses its environment
  - Fuses sensor data to form a model of the real world
  - Plans navigation paths
  - Actuates steering wheel, brake, and accelerator

However, ‘Boss’ was designed to
  - Win the 2007 DARPA Urban Challenge competition
  - Run for a few hours (60 miles in less than 6 hours)
Boss System Architecture

Motivation → SAFER → SysWeaver → Evaluation → Case Study → Conclusion
Task-Machine Mapping and Machine Failure

<table>
<thead>
<tr>
<th>Node 4</th>
<th>Node 5</th>
<th>Node 6</th>
<th>Node 7</th>
<th>Node 8</th>
<th>Node 9</th>
<th>Node 10</th>
<th>Node 11</th>
<th>Dev Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Planning</td>
<td>CAN Interface</td>
<td>Behavior</td>
<td>Sensor Fusion</td>
<td>Mission Planner</td>
<td>GPS Task</td>
<td>Motion Planning</td>
<td>Building Map</td>
<td>Perception Task for Velodyne</td>
</tr>
<tr>
<td>CAN Interface</td>
<td>Perception Task for SICK</td>
<td>Perception Task for SICK</td>
<td>Perception Task for IBEO</td>
<td>Sensor Fusion</td>
<td>Perception Task</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perception Task for ARS300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 10 boards each with a dual-core processor were used

**Fault Conditions**
- Node 12 started to fail relatively frequently
- The perception task for a critical 3-D sensor (Velodyne) would stop
  - Defensive Driving mode would be turned on: very slow progress

**Goal:** Autonomous driving functions should run normally and safely even if 1-2 processor boards fail.
Fault Models

- Handle permanent processor failures and permanent task failures
  - Tolerate a given number of failures
- Primary fault model: *fail-stop*
  - An entity stops functioning when it fails instead of alternating between correct and wrong outputs
- Fault-containment support must be provided
  - At OS, board and subsystem levels
Outline

- Motivation and Goals
- **SAFER**: System-level Architecture for Failure Evasion in Real-time Applications
- SysWeaver
- Evaluation and Case Study
- Conclusion
Trends in Autonomous Vehicles

Autonomy

Fully-autonomous

On-demand autonomous
- Automatic lane change
- Autonomous stop-and-go
- Self-driving on highway

Active safety
- Collision warning/avoidance
- Adaptive cruise control

2015

2020

Bring more features, but less resources...
Related Work

- Generic distributed fault-tolerant systems: ISIS, FT-CORBA, Arjuna, REL, IFLOW, etc.
  - Timely failure recovery is not the primary objective
  - Overhead is relatively large for autonomous vehicles
- Real-time fault-tolerant systems: MEAD and FLARe
  - Main focus is on soft real-time systems
  - The publish-subscribe model is not supported
- Fault-tolerant scheduling algorithms
  - can be an input to our framework
SAFER: System-level Architecture for Failure Evasion in Real-time applications

- On each CPU, SAFER monitors health status, stores state information of the primary and broadcasts the information
  - The primary-backup architecture is used
  - Two types of backups: hot standby and cold standby
  - If the primary and all hot standbys die, start cold standby with the transferred state information
  - Transmit state information on communications bus for other SAFER layers
  - Status manager and health monitor:
    - monitors/obtains the health status of sensors, actuators, power systems, processors, and communication links and
    - distributes any fault notifications
SAFER: System-level Architecture for Failure Evasion in Real-time applications
SAFER: Task Structure

- Using **passive replication**
  - Each task uses the publish-subscribe model for communications
  - Depending on the backup type, a backup task may always be running
    - Hot standby: SAFER **library** manages the necessary data to communicate
    - Cold standby: SAFER **daemon** manages the necessary data to transfer
SAFER: Fault Detection

- **Time-based failure detection**
  - Hot standby: Hot standbys keep track of the heartbeats from the primary
    - Tradeoff between fault-detection latency and bandwidth usage
  - Cold standby: The SAFER daemon running on the processor with the cold standby will detect the failure of the primary

- **Event-based failure detection**
  - The local SAFER daemon detects a task failure and triggers recovery immediately
    - The local SAFER daemon can capture the signal
    - In case of processor failure, this detection cannot be used
SAFER: Fault Recovery

- **Promote** a backup to the primary
  - Chosen automatically from available backups based on static information
    - such as node address
- **Re-spawn** the halted task
  - only possible when a task failure happens
SAFER: Hot Standby

- **Application group**
  - the primary (the master SAFER daemon) and
  - its hot standbys (the slave SAFER daemons)
SAFER: Primary Failure

- The primary can try to re-launch itself $n$ times (only for the task failure)
- If a hot standby misses heartbeat signals three times, the most preceding hot standby among the remaining will become the primary (according to the precedence values for each member)
- If the previous master rejoins, it will become the new master after demoting the current primary
SAFER: Cold Standby

- A cold standby will be launched only if the primary and its hot standbys are all dead
When a primary fails, the master SAFER daemon will detect the failure and send a message to promote the cold standby to become the primary.
Outline

- Motivation and Goals
- SAFER: System-level Architecture for Failure Evasion in Real-time Applications
- SysWeaver
- Evaluation and Case Study
- Conclusion
SysWeaver: Model-based Design Tool

- **System simulation**
  - Tasks
    - Simulation in tasks
      - Average-case
    - Worst-case execution times
  - Preemption of tasks
  - Primary-backup support
  - Fault-triggered mode change
  - Network
    - Simulation of network buses

- **Analysis output**
  - Recovery time of the primaries
  - Timeline of simulation results
System Schedulability Test for SysWeaver

Response time test for the primary

Response time test for the hot standbys

Response time test for the cold standbys

```
Input: \( \Gamma \): a taskset, \( P \): a set of processors and \( \Pi \): allocation information between \( \Gamma \) and \( P \)
Output: Schedulability of \( \Gamma \) on \( P \) with \( \Pi \)
1: for \( i = 1 \) to \( n \) do
2: \( \triangleright \) Do the response time test for \( \tau_i \) and its standbys
3: \( R_i \leftarrow \) the response time of \( \tau_i \) on \( \Pi_i \).
4: if \( R_i \leq D_i \) then
5: \( \triangleright \) The primary is schedulable, so check its standbys
6: for \( j = 1 \) to \( n^H(\tau_i) \) do
7: \( R_j \leftarrow \) the response time of \( \tau_{i,j}^H \) on \( \Pi_{i,j}^H \).
8: if \( R_j \leq D_i \) then
9: \( \triangleright \) This hot standby is schedulable.
10: \( \triangleright \) Check its slack-to-recovery time
11: if \( R_i \leq \mu_iD_i - kT_{\text{heartbeat}} - d \) then
12: \( \triangleright \) The primary and hot standbys are recoverable
13: for \( j = 1 \) to \( n^C(\tau_i) \) do
14: \( R_j \leftarrow \) the response time of \( \tau_{i,j}^C \) on \( \Pi_{i,j}^C \).
15: if \( R_j \leq D_i \) then
16: \( \triangleright \) This cold standby is schedulable.
17: \( \triangleright \) Check its slack-to-recovery time
18: if \( R_i \leq \mu_iD_i - R_j - kT_{\text{heartbeat}} - d - d_S \) then
19: \( \triangleright \) Mark this cold standby recoverable
20: if all tasks schedulable and recoverable then
21: return TRUE
22: else
23: return FALSE
```
Components contain Execution Time Distribution (Min, Max, Average)

Standby components can be configured as hot or cold
SysWeaver Simulation Timeline

5 ms

14 ms
Outline

- Motivation and Goals
- SAFER: System-level Architecture for Failure Evasion in Real-time Applications
- SysWeaver
- Evaluation and Case Study
- Conclusion
SAFER: Implementation

- Integrated into the existing Boss Software Architecture
  - Implemented on Ubuntu 10.04 x86
  - C++ is used
  - The existing APIs are preserved
  - SAFER can be configured through the configuration parameter interface
    - SAFER configuration can be modified without re-compilation
  - Each data point corresponds to an average of 50 iterations
Measurement Metric

- **Fault detection time:**
  - The time duration between
    - When a failure happens,
    - And when the failure is detected by
      - Hot standby for the primary having hot standbys
      - The local SAFER daemon for cold standby

- **Fault recovery time:**
  - The time duration between
    - When the failure is detected by
      - Hot standby for the primary having hot standbys
      - The local SAFER daemon for cold standby
    - And when the failed task is completely recovered
Time-based Failure Detection with Hot Standby

- Theoretical analysis from SysWeaver is shown
- The period of heartbeat signals affects the failure detection time
- The period of the task affects the failure recovery time
Time-based Failure Detection with Cold Standby

- Theoretical analysis from SysWeaver is shown
- Recovery time is increased significantly due to the initialization process
Event-based Failure Detection

- **10ms-period task**
  - Failure Detection Time
  - Failure Recovery Time
  - The Worst-Case

- **100ms-period task**
  - Failure Detection Time
  - Failure Recovery Time
  - The Worst-Case

- **Failure detection time is decreased significantly**
  - The local SAFER daemon detects a task failure and lets other daemons know immediately

- **This method *cannot* be used for processor failures**
Overview of Case Study on Boss

- A simulation scenario used for “Robot City” located in Pittsburgh 2 miles from the Carnegie Mellon campus
- A hot standby for BehaviorTask and one for MotionPlanningTask
- The velocity profiles measured from SimpleControllerTask
Task-Machine Mapping Revisited

Node 1: mission planning
- RNDF
- MDF
- Mission Planning
- Road Network
- Mission Plan
- Request
- Replan
- Mission State
- Pose
- Planning Graph
- Scenario
- Goal
- Planning State
- World State
- Static Map
- Dynamic Map
- Road Map
- sensor Health
- Behavior-Labeled RNDF
- Visibility
- Pose
- Static Map
- Road Map
- Dynamic Map
- World State
- Visibility
- Vehicle Commands

Node 2: behavior/planning
- Replicated

Node 3: replicated

Node 1: simulated tasks
- Hardware
  - Velodyne
  - Down Looking SICK
  - Horizontal SICK
  - Vehicle Sensor CAN Bridge
  - Applanix
  - Cameras
  - Health Monitoring
  - Vehicle Controller (DbW)

Perception & World Modelling

Mission Planning

Behavior Generation

Motion Planning

World State

Static Map

Dynamic Map

Road Map

Sensor Health

Visibility

Node 1: mission planning
- RNDF
- MDF
- Mission Planning
- Road Network
- Mission Plan
- Request
- Replan
- Mission State
- Pose
- Planning Graph
- Scenario
- Goal
- Planning State
- World State
- Static Map
- Dynamic Map
- Road Map
- sensor Health
- Behavior-Labeled RNDF
- Visibility
- Pose
- Static Map
- Road Map
- Dynamic Map
- World State
- Visibility
- Vehicle Commands

Node 2: behavior/planning
- Replicated

Node 3: replicated
Case Study Scenario

- The cluster will run the simulation of one scenario from the DARPA Urban Challenge
- Three laptops with one switch are used
  - Node 1 is connected to the user interface
  - All laptops are required for running the simulation
  - Node 2 will be disconnected
  - The simulation continues to run
  - Node 2 will rejoin
A Case Study on Autonomous Boss

- Velocity profiles of the autonomous vehicle on this map
A Case Study with a Failure Injection

- No visible difference is detectable from the velocity profiles

The **recovery time**: 40 ms for BehaviorTask and 55 ms for MotionPlanningTask
Outline

- Motivation and Goals
- SAFER: System-level Architecture for Failure Evasion in Real-time Applications
- SysWeaver
- Evaluation and Case Study
- Conclusion
Conclusion

**SAFER:** System-level Architecture for Failure Evasion in Real-time applications

- Offers fault-tolerance of distributed embedded real-time systems such as autonomous vehicles
- Supports the primary-backup approach with hot and/or cold standbys
- Includes a health monitor and status manager
- Failure detection time and failure recovery time are measured
  - For time-based failure detection:
    - Period of heartbeat signals affects the failure detection time
    - Period of the task affects the failure recovery time
  - For event-based failure detection:
    - Failure detection time is hugely decreased compared above
    - This method cannot be used for processor failures

- Integrated into our autonomous vehicle
  - Demonstrated in a case study
Thank you and Questions?

Junsung Kim: junsungk@cmu.edu