Realizing a Fault-tolerant Embedded Controller on Distributed Real-Time Systems

Junsung Kim*, Praful Puranik†, Ragunathan (Raj) Rajkumar*
* Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
† Electrical Engineering, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal, India
Email: *{junsungk, raj}@ece.cmu.edu, †puranik.praful@gmail.com

***Abstract*—Advances in real-time, embedded and distributed systems along with control and communication theory have catalyzed the rapid emergence of cyber-physical systems such as a self-driving car. The importance of fault-tolerance support on a cyber-physical system (CPS) has been greatly emphasized by recent research due to the nature of CPS that senses its surroundings, processes sensor data, and reacts using its actuators. In order to tackle this challenge, we proposed SAFER (System-level Architecture for Failure Evasion in Real-time Applications) in our previous work. SAFER is able to tolerate fail-stop processor and/or task failures for distributed embedded real-time systems. One of its limitations, however, is that SAFER is not capable of tolerating a failure of a processor with a dedicated connection to an actuator. This paper provides a method that relaxes this limitation by (1) deploying a small piece of hardware to avoid a dedicated connection between a processor and an actuator, (2) adding a software module that monitors and controls the hardware, and (3) enhancing the failure detection and recovery mechanisms of SAFER to support these changes. The detailed implementation and evaluation of the SAFER extension is on-going work.**

I. Introduction

As real-time, embedded and distributed systems along with control and communication theory flourish together, various applications such as planetary robots, aerospace vehicles, smart cars, medical devices and nuclear power plants on smart grids have been rapidly and extensively deployed. Such cyber-physical systems (CPS) are tightly linked to physical space. A CPS recurrently retrieves raw data from sensors, perceives its surroundings, and controls actuators to accomplish its missions and/or react to environmental changes. Therefore, meeting strict timing guarantees in CPS is of high importance [1].

System reliability is also an essential factor in CPS applications due to the interactions with the physical world. Although there has been much research on building reliable distributed embedded real-time systems [2, 3, 4, 5, 6, 7], a trend towards more complex features in a system with cost and space constraints poses challenges in developing a reliable system. For example, a recent high-end vehicle already has several active safety features such as adaptive cruise control, collision avoidance, lane departure warning, parking assist, etc. as depicted in Figure 1. Such a vehicle may not have enough space or it may become too expensive to deploy traditional hardware redundancy for all CPS features to meet reliability requirements. Higher assembly costs coming from extra hardware redundancy may not be desirable either. This trend is expected to continue as these features will be available in mid-range cars in the near future as well.

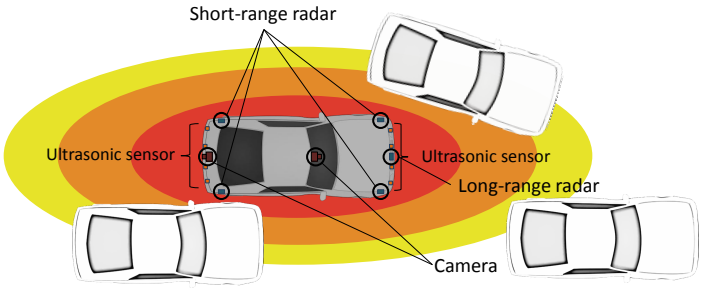

Fig. 1. A car can react to various situations using different types of sensors and active safety features. This is an exemplary sensor configuration. [8]

In order to address these challenges, we proposed SAFER (System-level Architecture for Failure Evasion in Real-time applications) in our previous work [9, 10]. SAFER is a distributed software layer enabling *task-level* fault-tolerance features on distributed embedded real-time systems. SAFER is capable of tolerating *fail-stop* processor and/or task failures. SAFER has a health-monitoring module per daemon running on each node. Each daemon subscribes to health status and state information to detect failures. When a failure is detected, SAFER prevents a system failure from happening by using task-level hot standbys, cold standbys and re-execution. The state information is accordingly leveraged to meet recovery-time requirements.

SAFER provides an adaptive and affordable way of tolerating processor and/or task failures on distributed real-time embedded systems. However, it has a limitation in tolerating a failure of processor that has a dedicated connection to an actuator,[1] which plays an essential role in CPS applications. SAFER uses the publish-subscribe model for data transfer among all system nodes. Any node connected to a network including an actuator is recoverable using SAFER. Even though distributed control on a publish-subscribe network is getting popular [11, 12], most actuators require separate I/O connections for enabling, controlling and disabling themselves. For instance, many electric motors receive direct pulse-code modulation (PCM) signals from analog input ports to control dynamics. Hence, it is common to have a separate embedded controller for such a motor. Then, the failure of this controller itself cannot be tolerated by SAFER because those signals are

Praful Puranik contributed to this work while he was still at Carnegie Mellon as a summer intern.

[1] A failure of processor having a direct connection to a sensor cannot be tolerated, neither. Although this paper focuses more on an actuator side, a similar approach can be used for a processor dedicatedly connected to a sensor. We leave it as future work.

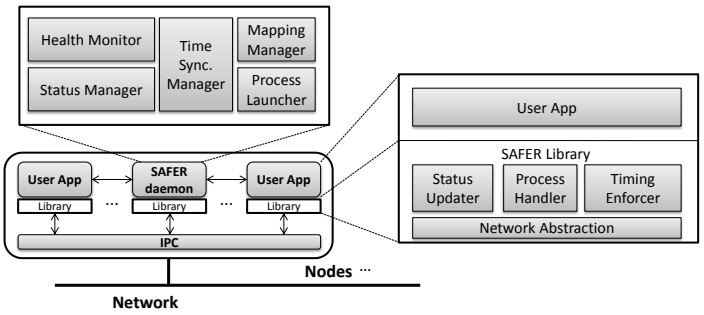

Fig. 2. The overall architecture of SAFER

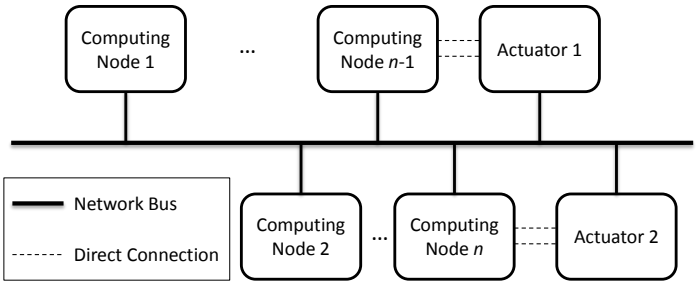

Fig. 3. An example of actuator connections on SAFER

not connected to the network. In other words, the controller becomes a single point of failure.

This paper provides a method that relaxes this limitation by (1) deploying a simple piece of hardware to avoid a dedicated connection between a processor and an actuator, (2) adding a software module that monitors and controls the hardware, and (3) enhancing the failure detection and recovery mechanisms of SAFER to support these changes. The detailed implementation of the SAFER extension is the subject of on-going work.

The rest of this paper is organized as follows. Section II outlines SAFER and describes actuator connections on SAFER. Section III describes the proposed method and the corresponding changes that need be made to SAFER. We provide a brief overview of related work in Section IV, and we summarize our work and discuss future work in Section V.

II. A Distributed Layer for Task-level Fault-tolerance and its Actuator Connections

In this section, we summarize how SAFER works and how actuators are connected. More detailed information about SAFER can be found in [10].

A. SAFER Architecture Overview

On a system composed of n computing units connected via network such as Ethernet or Control Area Network (CAN) [13], SAFER provides configurable fault-tolerance features: task-level hot/cold standby and re-execution. The overall architecture of SAFER is depicted in Figure 2 [10]. The SAFER layer is composed of one daemon per computing node, and a task library. The daemon on a node is responsible for governing tasks on each node and monitoring the health and state information of the node and its tasks. The library enforces the timing requirements of each task; SAFER guarantees the periodic behavior of each task using a reservation-based kernel module Linux/RK [14]. When a task is launched, the SAFER library configures the task by reading the control parameters stored on a disk. Tasks on SAFER communicate using a publish-subscribe model for inter-process communications [15, 16].

The status updater of the SAFER library shown in Figure 2 broadcasts heartbeat signals for detecting task/processor failures. When there are a few consecutive missing signals, the standby node declares the failure of the primary. The SAFER daemon can also detect a task failure by capturing an OS signal generated when an unexpected task failure happens. We call these two different schemes as *time-based* and *event-based* failure detections, respectively. Once a failure is detected, a proper recovery procedure based on the standby type is triggered. If the type is a hot standby, one of the hot standbys is simply promoted to the primary. If the type is a cold standby, the standby will be launched with the stored state information that is regularly broadcast with heartbeat signals. The state information is also used when a task re-execution occurs on an event-based failure detection. The worst-case response time analysis of failure detection and recovery time on SAFER was also discussed in [10], and an analysis engine was integrated into a model-based design tool called SysWeaver [17]. SysWeaver can help application developers to choose desired parameters at the design time.

The SAFER layer is implemented on Ubuntu 10.04.4 LTS and integrated into an autonomous vehicle Boss [18] developed at CMU. A case study on the vehicle empirically showed that the failure detection and recovery procedures operated by SAFER do not cause any noticeable difference at the level of autonomous driving [10].

B. Actuator Connections on SAFER

In this subsection, we will describe how actuators are used with SAFER taking the Boss implementation as an example. SAFER assumes the publish-subscribe model for data delivery among all computing units. Since SAFER is capable of tolerating a task-level failure, the failure of any running task with standby(s) connected to a network can be recovered from. On SAFER, an actuator can also subscribe to its publishers that control the actuator. This architecture is becoming common for distributed control [11], and it can be a desirable way to move forward [12]. For example, all motors that control the steering, acceleration, brake and gear shift of Boss are connected to a CAN bus, and all control messages are transferred via the bus.

A problem, however, arises when an actuator requires dedicated I/O connections. For instance, all electric motor units used on Boss receive three individual direct inputs and one output in addition to the control messages from CAN. The inputs are used for enabling/disabling the motor and controlling its rotation. The output represents the current status of the motor. A high-level connection diagram of Boss is depicted in Figure 3. From the figure, Actuator 1 has direct connections to Computing Node $n-1$, and Actuator 2 also has direct connections to Computing Node n. Then, (say) the failure of Computing Node n may cause a single point of failure because Actuator 2 cannot be used anymore[2]. This cannot be resolved

[2]Boss is designed to be *fail-safe* when this happens.

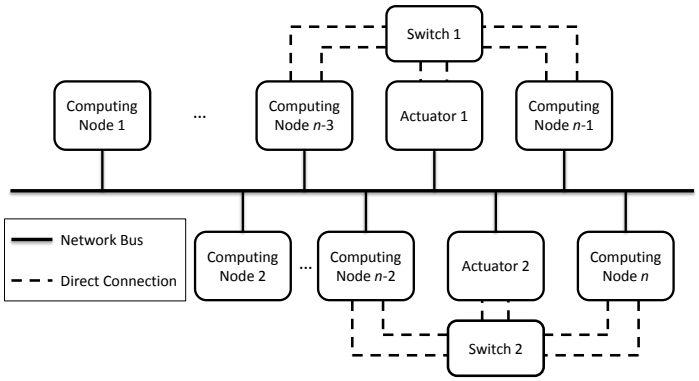

Fig. 4. An example of actuator connections with switches on SAFER

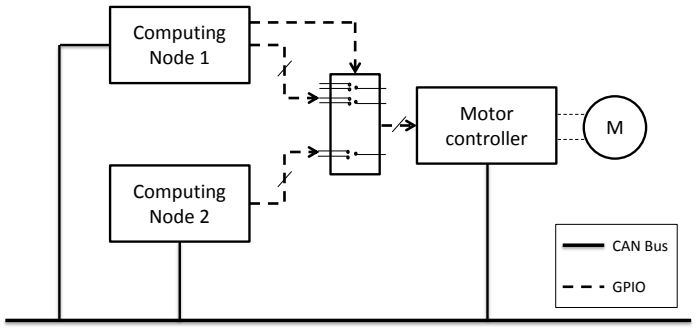

Fig. 5. A detailed switch diagram with the primary and standby nodes

even though SAFER is enabled and Actuator 2 does not have its own fault.

III. The Proposed Method

We address the problem stated in the previous section by adding a simple switch, designing a new software module for controlling the switch and enhancing the SAFER layer to support these changes.

A. Switch Design and its Control

The main idea of the proposed solution is that we do not allow any direct connection between an actuator and its controlling unit. In other words, we enable more than one computing unit to be able to control any actuator. An actuator unit usually has several input and output connections. Any number of computing units can be connected to the output ports from the actuator unit. This resembles a publish-subscribe model, and several computing units can monitor the output of the actuator unit using a relatively simple circuit. The challenge arises from the input side of the actuator unit because only one unit should control the actuator at any given time. To satisfy this requirement, we need to have an additional switch which multiplies one and many possible inputs (outputs from controlling CPUs going to the actuator) and enables only one set of inputs to the actuator. This architecture is illustrated in Figure 4 when a computing unit for an actuator has one standby[3].

The reader may note that this switch itself is a single point of failure, but in practice a simpler unit can be built to have less probability of failure than a complex unit. Let p denote the probability of failure for a switch. Let q denote the probability of failure for a computing unit. We assume that (1) actuators do not fail and (2) all tasks have standbys so that they can tolerate two processor failures. Suppose $p << q << 1$. This is a reasonable assumption because a computing unit on a CPS usually has a probability of failure that is much less than 1. Since the switch is supposed to be a very simple circuit, p is even less than q. Then, from the architecture in Figure 3, the probability of system failure is $2q$. For the architecture using the switch from Figure 4, the probability is $2(q^2 + p)$. Based on the assumption $p << q << 1$, $2(q^2 + p) << 2q$ satisfies, and the switch architecture from Figure 4 is significantly better in terms of reliability.

[3]The number of standbys affects the switch design.

Figure 5 shows a more detailed diagram focusing on a primary computing unit (Computing Node 1), its standby (Computing Node 2) and an actuator unit (Motor Controller and Motor). For the switch design, we decided to use a relay due to its simplicity and reliability. As shown in the figure, the connection is controlled by the primary. When the standby fails, the primary is still in place. When the primary fails, its standby will take over. The switch can be designed to automatically make connections between the actuator and the standby when the control signal from the primary is absent. A daisy chain configuration can be used to support multiple standbys. Thus, we can tolerate a failure of a node that controls an actuator that requires direct input and output connections.

B. SAFER Modifications

SAFER needs to be modified to support the above-mentioned switch. Two main modifications are required: (1) the switch added in Figure 5 should be supported by the SAFER layer; and (2) the failure detection and recovery scheme should be extended.

SAFER must control connections between an actuator unit and its computing units based on which node is the primary. In other words, when the primary takes control, the relay should maintain the connection between an actuator and the primary. If the standby is promoted to the primary due to the failure of its primary, the relay should alternate the connection. This operation can be done on SAFER by modifying appropriate software modules. For hot standby as the type of standby, the status updater of the SAFER library needs be modified to be able to set a GPIO pin[4]. It is important to promptly control the switch when the primary failure is detected. For cold standby as the standby type, this should be managed by the SAFER daemon. Therefore, the status manager of the SAFER daemon should manage the switch status, and the process launcher of the daemon should handle the switch when it activates the cold standby.

The SAFER library was originally implemented to subscribe to other tasks via an inter-process communication primitive using Ethernet. Currently, it can send heartbeat signals using only one communication interface, and this is also a limitation. For example, Boss has a few controllers only on CAN. To make them work together with computing units on an Ethernet, the network abstraction should be modified to

[4]This is for the current implementation. Depending on how SAFER is implemented, the modification may be different.

support two or more different network types at the same time. Then, a standby on the Ethernet can subscribe to the heartbeat signals of the primary on CAN. In this case, additional care must be given to time-based failure detection because the network characteristics are different between Ethernet and CAN. The failure recovery scheme also requires more design-time analysis. The original SAFER assumes all binaries of the various standbys are the same as the primary's. However, when a different network is used, the binary itself and configuration parameters can be different, hence making design-time analysis more difficult.

IV. Related Work

Building fault-tolerant distributed embedded systems has a long history, and there has been a significant amount of research [5, 4]. In particular, CORBA-based fault-tolerant research is well summarized in [19]. There has also been extensive research in the real-time systems community for supporting fault-tolerance features [2, 3, 6, 7]. This series of research has been trying to provide timely failure detection and recovery. Our previous research [9, 10] has been studied to (1) support predictable timing characteristics of failure detection and recovery, (2) provide an adaptive infrastructure for failure detection and recovery, and (3) work on a publish-subscribe model for flexible sensor/actuator usage. On top of this, the paper addresses how the failure of a processor that has a direct connection to an actuator can be tolerated. There has also been research on building a reliable distributed control system [20, 11, 21], which can be complementary to our research.

V. Summary and Future Work

We have proposed a method to realize a fault-tolerant embedded controller on distributed real-time systems. We have avoided a direct connection between an actuator and a computing unit so that the actuator can be controlled by other computing units in the distributed real-time system. To apply this idea, we have proposed a way to modify SAFER (System-level Architecture for Failure Evasion in Real-time applications). We achieve this goal by (1) adding simple relay circuits controlling the connections between actuators and computing units, (2) adding a software module for maintaining the relay circuits, and (3) enhancing the failure detection and recovery schemes of SAFER. The hardware design has been completed, but the implementation of the SAFER extension is on-going.

Future work to be done is to complete the implementation so that SAFER can work with the additional hardware. For failure detection and recovery, the network abstraction of the SAFER library should also be modified as stated in Section III. We will evaluate its performance in terms of failure detection and recovery time. Also, the analysis engine on SysWeaver should be modified for the design-time analysis accordingly.

References

[1] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, pages 731–736. ACM, 2010.

[2] J. Balasubramanian, et al. Middleware for resource-aware deployment and configuration of fault-tolerant real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010.

[3] J. Balasubramanian, et al. Adaptive failover for real-time middleware with passive replication. In *Proc. of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

[4] Object Management Group. Fault-Tolerant CORBA. *OMG Technical Committee Document formal/2001-09-29*, September 2001.

[5] K. Birman. Replication and fault-tolerance in the isis system. In *Proc. of the 10th ACM symposium on operating systems principles*, 1985.

[6] P. Narasimhan, et al. MEAD: support for Real-Time Fault-Tolerant CORBA. *Concurrency and Computation: Practice and Experience*, 17(12):1527–1545, 2005.

[7] P. Emberson and I. Bate. Extending a task allocation algorithm for graceful degradation of Real-Time distributed embedded systems. In *Real-Time Systems Symposium, 2008*, pages 270–279, 2008.

[8] GM Press. Sensor Fusion Enables Cadillac Safety Advancements.

[9] J. Kim, R. Rajkumar, and M. Jochim. SAFER: System-level Architecture for Failure Evasion in Real-time Applications. In *Proc. of the 4th Workshop on Adaptive and Reconfigurable Embedded Systems*, 2012.

[10] J. Kim, G. Bhatia, R. Rajkumar, and M. Jochim. SAFER: System-level Architecture for Failure Evasion in Real-time Applications. In *Proc. of the 33rd IEEE Real-Time Systems Symposium (RTSS)*, 2012.

[11] W. Xiang, P.C. Richardson, C. Zhao, and S. Mohammad. Automobile brake-by-wire control system design and analysis. *Vehicular Technology, IEEE Transactions on*, 57(1):138–145, 2008.

[12] B.S. Heck, L.M. Wills, and G.J. Vachtsevanos. Software technology for implementing reusable, distributed control systems. *Applications of Intelligent Control to Engineering Systems*, pages 267–293, 2009.

[13] Bosch. *CAN Specification version 2.0*. Bosch, 1991.

[14] S. Oikawa, et al. Portable rk: a portable resource kernel for guaranteed and enforced timing behavior. In *Proc. of the 5th IEEE Real-Time and Embedded Technology and Applications Symposium*, 1999.

[15] M. McNaughton, et al. Software infrastructure for an autonomous ground vehicle. *Journal of Aerospace Computing, Information, and Communication*, 5(1):491 – 505, December 2008.

[16] C. Urmson. SimpleComms. http://www.cs.cmu.edu/~curmson/SimpleComms.tgz as of Jan 10, 2013.

[17] D. de Niz, et al. Model-based development of embedded systems: The sysweaver approach. In *Proc. of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[18] C. Urmson, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(1):425–466, June 2008.

[19] S.M. Sadjadi and P.K. McKinley. A survey of adaptive middleware. *Michigan State University Report MSU-CSE-03-35*, 2003.

[20] R. Isermann, R. Schwarz, and S. Stolzl. Fault-tolerant drive-by-wire systems. *Control Systems, IEEE*, 22(5):64–81, 2002.

[21] P. Sinha. Architectural design and reliability analysis of a fail-operational brake-by-wire system from iso 26262 perspectives. *Reliability Engineering & System Safety*, 96(10):1349–1359, 2011.