Using Adaptive Fault-Tolerance for Automotive and Aerospace Applications

Y. Xe†, I. Koren†, E. M. Atkins‡, C. M. Krishna†, and K. G. Shin‡
† University of Massachusetts, Amherst, MA
‡ University of Michigan, Ann Arbor, MI

I. Introduction

There is a trend in transportation applications such as automobiles and aerospace vehicles towards integrating control functions in a common (share) computing platform rather than having multiple computers each handling a specialized function. Thus, instead of having each subsystem having its own dedicated processors, we have a set of processors that can be used for whichever function requires their services. This approach allows for pooling of resources and adjustment of resource allocation based on the changing urgency and extent of computational resource requirements for the various vehicle control tasks. It can take advantage of the increased processing power available in today's processors. Autonomy of the control system requires extensive fault-tolerance; fault-tolerance requires massive redundancy which can be expensive, both to provision and to run.

In this position paper, we argue that conventional fault-tolerance in ultra-reliable applications like ground and air/space vehicles is usually quite wasteful and that it is possible, using dynamically adjustable fault-tolerance, to provide the same level of safety while dramatically reducing the workload stress imposed on the computers. Since workload is correlated with failure rates, this has the potential to reduce the system failure rate as well.

II. Basics of Adaptive Fault-Tolerance

The extent of fault-tolerance provided to any task should be related to the requirements of the controlled system at that point. Much, if not most, of the time, the controlled plant is deep within its allowed state space, and can safely tolerate certain types of errors in the control computer. During such times, the level of fault-tolerance can be reduced. Such a reduction allows the computational resources to be used for other tasks or for the workload to be reduced. The former can improve the quality of service provided (including service to less essential tasks); the latter reduces the stress on the computational subsystem, thereby reducing its failure rate. Further, it may be possible in certain cases to be more relaxed about letting the computer miss a few deadlines, which again reduces the computational workload.

[0]Supported in part by the National Science Foundation under grants CNS-0930813, 0931035, 1329702, and 1329831.

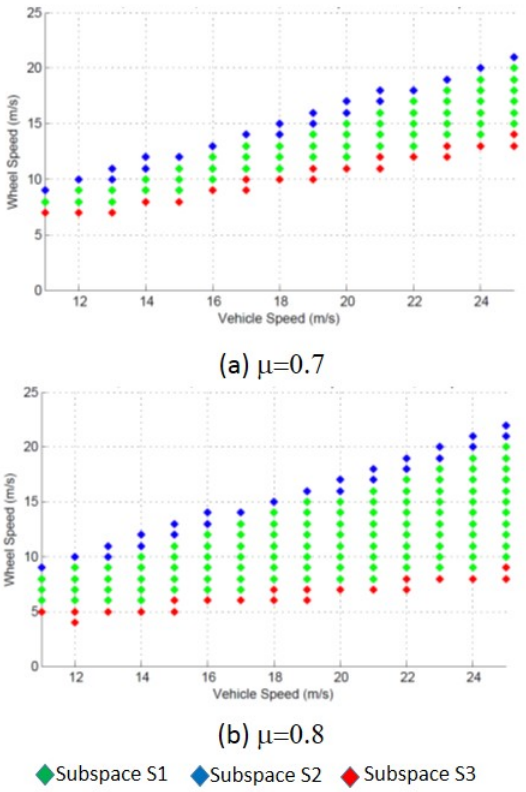
Some prior work exists on adaptive fault-tolerance; see, for example, [1], [3]–[5].

The state-space can be divided into three parts as described and illustrated in [2]: S_1 in which the system can tolerate any wrong actuator output, S_2 requiring either a correct output or no output at all (fail-zero), and S_3, requiring full fault-tolerance (where a correct output is essential). S_1 is so deep within the allowed state-space that it is possible for the computer to cause the actuators to be set in the worst-case wrong configuration without the vehicle leaving the safe state-space. Obviously, a sufficiently long succession of incorrect outputs can cause catastrophic failure; this subdivision only applies to the output associated with a single control-task iteration. S_1 requires no fault-tolerance: a single copy of the task can be run and its output sent to the actuator(s). S_2 is somewhat closer to the edge of the allowed state-space; the fail-zero property is typically generated by having a duplex of processors running the control task in parallel and requiring unanimity to produce the output. If no output is forthcoming (because the processor outputs disagree), the actuator(s) can be left in their previous configuration. Finally, S_3 is so close to the edge of the allowed state-space that even a single incorrect iteration can render the vehicle unsafe; in such a case, forward error-masking techniques such as triplexes with voting can be used.

III. Examples

A. Anti-Lock Braking System (ABS)

Suppose a car is braking to avoid hitting an obstacle a given distance away. Based on its current state (velocity, road conditions, and distance to the obstacle), it will be in one of these subspaces. If it is far from the obstacle,



Vehicle is 55 m from obstacle

Fig. 1: Subspaces for Vehicle Braking

its speed is low, and the road is dry, then no failure of the ABS system over one of its iterations can harm the vehicle; in such a case, it is sufficient to use just one copy and no fault-tolerance. If, on the other hand, the car is on a slippery surface, is operating at a relatively high speed and is not far from the obstacle it needs to avoid, then a malfunction for even one iteration is sufficient to place the vehicle in danger; in such a case, full forward error-masking must be allowed for. In between these two extremes, simple error detection (but no correction) is sufficient.

Figure 1 shows some simulation results. The vehicle is initially traveling in a straight line when an obstacle is detected a given distance away. Based on the initial wheel and vehicle speeds (the two are not always equal due to slip) and the condition of the road surface (and thus the friction that can be exploited in braking), the three subspaces are shown for two frictional coefficients: $\mu = 0.7$ and $\mu = 0.8$. As μ increases, i.e., the road surface becomes more benign, a greater fraction of the space is in S_1. In this preliminary example, we have assumed uniform road friction for all four tires; similar work can be done for a more complex road-car interaction. Likewise, we have not considered the interaction of steering with braking which can be studied along similar lines.

B. Sense-and-Avoid in Aircraft (SAA)

SAA is another example application that can use adaptive fault-tolerance. SAA involves three steps: (a) Predict (sense) a loss-of-separation condition, (2) warn the flight crew (ground controllers if unmanned; pilots if manned) and potentially compute a maneuver to prevent loss-of-separation and (3) execute an avoidance maneuver to prevent collision. Once again, based on the current state of the aircraft and the one it is trying to avoid hitting, the SAA application will be in one of these three subspaces. Initially, when the separation is small, high fault-tolerance will be required; as avoidance maneuvers are carried out and the separation increases, less (or no) fault-tolerance will be sufficient for the collision-avoidance task. (Note that each task can have its own distinct fault-tolerance requirement; it is entirely possible for the collision-avoidance task to be run with low fault-tolerance while having to run another task involving, say, the stability of the aircraft, at high fault-tolerance.)

IV. Implementation

We are developing a system which accepts as input the state equations of the controlled plant, a model of the environment, and limits on the actuator configurations and force, and carries out a simulation to output the three subspaces. These subspaces will be stored as lookup tables that the controller operating system can use to determine the appropriate level of fault-tolerance. An important issue is to be able to store these lookup tables compactly. How to do this reliably is the focus of current work. Also, note that when the vehicle in S_1 or S_2, the system has the option of dropping an iteration altogether if it is currently under a heavy workload; this is the approach shown in [6] to significantly improve the schedulability of task sets under heavy loading.

References

[1] S. Bak, K.K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety," *IEEE Real-Time and Embedded Technology and Applications Symposium* (RTAS), 2009.

[2] C.M. Krishna, "State-Based Fault-tolerance for Control Computes in Cyber-Physical Systems," *submitted for publication*.

[3] J. Goldberg, I.B. Greenberg, and T.F. Lawrence, "Adaptive Fault Tolerance," *IEEE Workshop on Advances in Parallel and Distributed Systems*, 1003, pp. 127–132.

[4] Z.T. Kalbarczyk, R.K. Iyer, S. Bagchi, and K. Whisnant, "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 6, June 1999, pp. 560–579.

[5] K.H. Kim and T. Lawrence, "Adaptive Fault-Tolerance in Complex Real-Time Distributed Applications," *Computer Communication*, Vol. 15, No. 4, May 1992, pp. 243–251.

[6] J. Lee and K. G. Shin, "Development and Use of a New Task Model for Cyber-Physical Systems," *submitted for publication*.