Distributed Control Architectures: New Middleware for Smart Software and Hardware Scheduling

A keynote presented to the Science and Technology Organization of the
North Atlantic Treaty Organization
On May 11, 2021

Eric Feron
Professor of Electrical Computer, and Mechanical Engineering
King Abdullah University of Science and Technology
eric.feron@kaust.edu.sa

Dear listeners, the point of my presentation is to propose the use of intelligent algorithms to support the management of safety-critical software. Such software includes, of course, the embedded software that implements engine control architectures.

There are two types of computing machinery. On the one hand, there are monolithic machines that are highly reliable and massive. Such machinery is very costly and it is present in all of today's systems wherever such systems are safety-critical. That includes all aircraft and their subsystems, including their propulsion systems. On the other hand, we are witnessing the ubiquitous emergence of very powerful and miniaturized computing systems. These computing systems are present in our cellular phones and our laptop computers. While they are much less reliable than their safety-critical counterparts, they are also far less expensive. The popularity of the devices they power means that far more investment goes into R&D and manufacturing of these devices rather than into the devices that our industry has grown accustomed to. The split between consumer-oriented computing systems and the safety-critical computing systems we rely upon means that our industry is at risk of being left behind by the rapid evolution of technology. What is even worse, our system development paradigms and assumptions may become invalidated if we cannot procure the chips our development methods for safety-critical systems have grown to rely upon: What is the worth of years developing regulations and "good practices" in embedded computer architecture development if the only chips we can find in the future are multi-core processors that we do not understand? What is the worth of the many SAE documents, ARP 4761 but especially ARP 4754, and what to think of the RTCA documents, such as DO-178C or DO-254 if the practices they refer to disappear because the hardware upon which they rely has gone? Our industry is facing the necessity to adapt to the rapid evolution of the hardware that supports its safety-critical information technology. Today, there are groups that work very hard on incorporating multi-core systems and distributed computing within aerospace standards. For example, the Certification Authorities Software Team – CAST – has produced a document, CAST – 32 aimed at exploring and defining the proper use of multicore processors inside embedded safety-critical systems. The purpose of such activities is to see whether we can adopt better practices to leverage multi-core processors than, say, waste the multi-processor opportunity to accommodate our habits by turning off all cores but one.

With the support of SAFRAN and the US Air Force Research Laboratory, my group has embarked into the process of doing just that: Try and make the best use of available and increasingly distributed and powerful computational hardware. We have explored several directions, which I will now describe in more detail.

First, there is the opportunity to use more distributed computational procedures for the purpose of reducing the acquisition cost and maintenance of embedded control architectures for propulsion systems. Here, we are interested in the traditional, point-to-point control architecture used in jet engines, whereby a centralized computer performs several missions, drawing information, such as temperature, pressure, angular speeds directly from available sensors and sending the result of its computations to the appropriate actuators, such as air bleed vanes, fuel flow valves, or ducts. This centralized architecture offers strong control over event timing and the possibility of completely mastering the information flow process and chasing bugs until no frames are dropped and the entire system works perfectly. Such an architecture is the one I chose, together with my MIT student team when we embarked into building the avionics of what would end up being the first fully autonomous, aerobatic helicopter in the world back in 2001. In the context of propulsion systems, I heard accounts of other bonuses offered by point-to-point architectures, which is the presence of many cables used by the embedded computing system to communicate with sensors and actuators offer as many grip points for the people in charge of monitoring and maintaining these engines. A distributed control architecture transforms this legacy system into an architecture based on a data bus, whereby each component of the system connects to the data bus and the bus is in charge of conveying the information around all sensors, actuators, and computers. This distributed architecture offers the advantage of improved flexibility, allowing each element to be replaced or updated by another and use a standardized communication medium to ensure all components can still talk to each other. However, the basic safety-criticality issue remains and, while a lot flexibility is enabled by a distributed computing network, the quality of the service delivered by the system must remain. For example, there may be more flexibility about when a packet or a frame may be sent out, but that packet should still be received and not accidentally dropped. During our research, we have found that significant control over activity scheduling over a communication network could be achieved by relying on dedicated optimization techniques originally developed by operations research for desktop applications. Such techniques include mixed-integer programming techniques.

Second, there is the opportunity brought by having a massive number of low-cost processors. While many think of such a computing bounty as the opportunity to cram in ever computationally greedier applications of signal processing, data mining and other prognosis and health monitoring products, my focus remains stuck on a single number that is 1e-9. This is the maximum number of lethal accidents a commercial airliner is allowed to experience every hour. Redundancy is one of the key and well-understood mechanisms that allows our trade to meet such an exorbitant reliability goal with what are usually far less reliable components, and redundant architectures have been part of the standard diet of safety-critical avionics engineers for as long as human lives have begun to matter in aviation. The new element we have addressed is understanding how we can address demanding redundancy needs when individual computing elements become frankly unreliable. That there be more of these elements is a given, but allocating tasks to them, whether the elements are neatly arranged in a multi-core architecture or scattered around with creative wiring options is not as obvious. In our laboratory, we have leveraged existing intelligent algorithms to perform highly flexible software allocation and re-allocation. Again, for us, that means using the optimization techniques I know best, including mixed-integer programming with the special twist that they must be used in a high-tempo environment. Armed with this philosophy, we have demonstrated how it is possible to take a large number of computing resources and dynamically re-allocate safety-critical tasks as computing elements fail, while making these failures invisible to the propulsion system operator. On the way to this demonstration, we have met interesting questions, such as "who allocates the software allocators?" This kind of questions is similar in nature to the question "who compiles the compiler?" familiar to computer scientists and is always the opportunity fer exchanging existential questions among researchers. Fortunately, the allocator allocation problem is well-posed: The allocator does indeed allocate itself. More precisely, three allocators decide who should be reallocated and who should perform the reallocation, should this become necessary. Related and important problems we got interested in along the way include developing reliable mechanisms to identify faulty nodes without saturating an already busy communication network.

Last, I would like to talk about distributed architecture design. Once again, the key number to remember is 1e-9. Given a set of low-cost computing resources, how do we connect them so that the service they provide can be performed with super-high reliability? Before going in this direction any further, I would like to point out that the question initially asked extends to the entire system design problem: Safety-critical functions also involve sensors and actuators. Therefore, a more appropriate question is: Given a set of computing resources, a set of sensors and a set of actuators, how do we connect them so that the service they provide is super-reliable? For example, my team currently aims at building a new kind of drone that features 256 brushless motors and 64 Pixhawk boxes, which include both sensing and computing capabilities. Finding out how I should wire all these components to obtain maximum vehicle reliability is a question that also makes sense. To answer these questions, we have developed off-line optimization models that rely on recently developed algorithms to solve geometric and signomial optimization problems. These optimization models allow us to answer questions such as, given a set of components, how reliable an architecture can we build based on these components? Or, given a reliability goal and a number of component options, what is the minimum price to be paid to achieve that reliability level? We have posed and brought satisfactory answers to these questions for both few and large numbers of components. We have also matched and validated our results against existing distributed computing architectures found on large airliners. Of course, when we face these problems, the data that is often missing is component-level reliability figures. Our discussions with aircraft manufacturers indicate that the main source of well-documented hardware is that provided by the automotive industry, thus indicating a possible rapprochement between the aerospace and automotive distributed computation systems in the future.