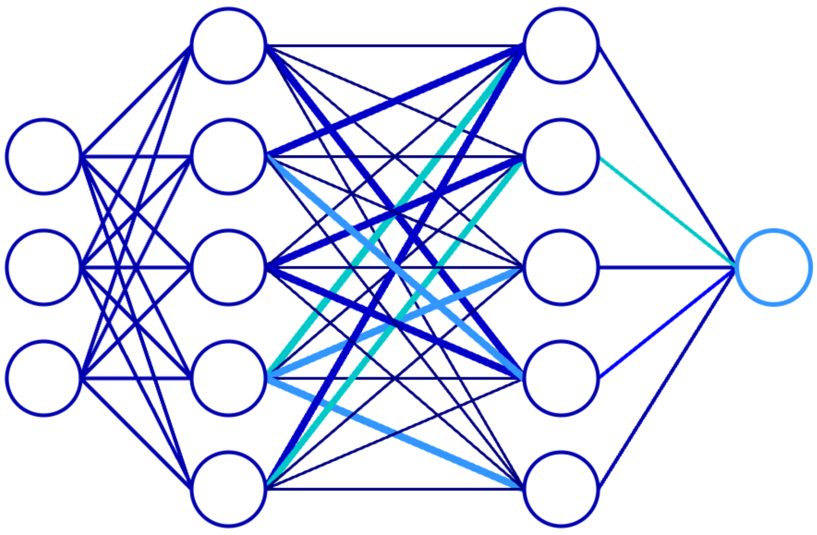

Real-Time Embedded Systems: A Quick Primer For AI

Susanna Cox | 28 November 2024 | susanna@anglesofattack.io

What Are Embedded Systems?

An embedded system is a computer system that has a dedicated purpose, within a larger electronic or mechanical system [1]. “Computer system” in this context means a processor, memory, and any peripheral I/O (input/output) devices that help the system do its job.

A key to understanding embedded systems is specificity. Embedded systems are designed to do a specific job, and while they might work at varying levels of complexity, it’s that dedicated functionality that makes them embedded.

General computing is the opposite of embedded systems. Most consumer computers (like laptops) are examples of general computing–they are designed to perform many diverse tasks. They have to do so reliably, or consumers won’t like their experience. As an example, laptops mostly do not need the level of specialization found in embedded systems to function and meet user expectations; and in fact such specialization would make general functionality impossible. (It may also be worth noting that consumer computers also contain embedded systems working within to enable all the processes and applications these machines run.)

Embedded systems can be very simple, or very complex, with a number of peripheral systems & devices working in concert. They are found in surprising places, from media players to the antilock brakes in cars to aircraft flight controls.

What makes a system embedded, regardless of the application, is dedication to a specific purpose. This gives rise to a more concise definition for an embedded system: a compute node that provides a *specific service* by processing inputs & returning responses, as part of a larger system.

Some embedded systems are also what are called real-time systems–systems that operate with additional timing constraints. Compute nodes that provide a specific service as part of a larger system, subject to strict timing deadlines, are called *real-time embedded systems* [1].

What Makes A System Real-Time?

Real-time systems are systems that operate within timing constraints, aka deadlines.

Real-time systems are subject to real-time constraints–in other words, they must deliver responses within specific time periods or by specific deadlines, depending on the system [2]. Crucially, this response timing has to be guaranteed.

Adding the time constraint to our definition from above, we can come to a more complete understanding: Real-time embedded systems are compute nodes providing a *specific service* by processing inputs and returning responses which are *guaranteed* within a predetermined timing scheme. The timing guarantee is critical to what makes a system real-time.

It’s easy to see why operating with time guarantees is vital to critical systems: as an example, consider the software that powers aircraft avionics. In the avionics use case, if signals are not received, processed, and sent back out quickly, critical flight functions might fail. Real-time functionality of these systems is necessary for flight safety. Failure to operate within deadlines could result in loss of life.

Real-Time Vs Fast

A common misconception about real-time systems, even among some engineers, is that the criteria for the category is based on speed–how fast a system can run [3]. But that’s not the case.

What is not necessarily widely understood, even among many engineers, is the degree to which computer behavior is non-deterministic. Computers are temperamental machines. If you’ve ever used a computer and seen it behave in an unexpected manner, you probably know this intuitively: different runs of programs can have different results. They also can–and do–take different lengths of time to complete seemingly identical processes.

Fast Vs *Reliable*

For this reason, when we are talking about computers and code, calculations for various processes often include minimum and maximum speeds, averages, or estimates. This brings to light a critical distinction: *speed vs reliability.*

Although real-time systems require rapid responses, they also require that the timing of these responses operates at microsecond precision levels. The variance of the latency–the wait between processes or signals–must be consistent. Latency variations between program iterations are so common and well-known, they have their own name: *compute jitter* [4].

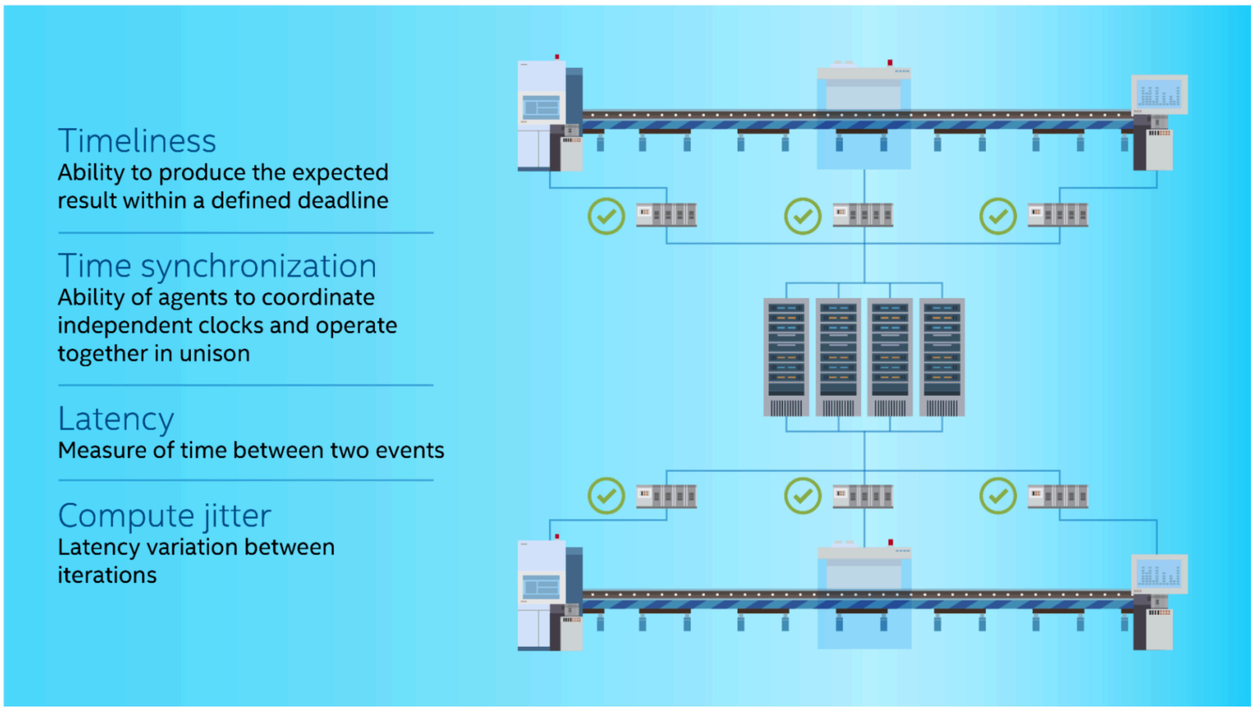
Timeliness
Ability to produce the expected result within a defined deadline
Time synchronization
Ability of agents to coordinate independent clocks and operate together in unison
Latency
Measure of time between two events
Compute jitter
Latency variation between iterations

*Image 1: Timeliness, Syncronization, Latency, and Compute Jitter. Source: **Intel*** [4]

Real-time is not so much about being fast, it's about being predictable. Predictable latencies, and predictable order of signals, all increase reliability.

Obviously speed is necessary, but safety critical systems demand more: precision timing. In fact, deterministic–also known as predictable–behavior is a crucial facet of real-time system operation:

"*A real-time computer system can be defined as a system that performs its functions and responds to external, asynchronous events within a predictable (or deterministic) amount of time.*" [5]

Put another way, in a real-time system, the outcome of an operation is equally important as the timing:

"*In a real-time system the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced.*" [6]

In systems where there is no tolerance for error, fast isn't good enough.

Not all real-time systems are safety-critical, however. Within the real-time category, there are different applications with different levels of response timing stringency.

Hard Vs Soft Real-Time

An important defining characteristic of hard vs soft real-time systems are the consequences for failure. Hard real-time systems are often safety-critical, with catastrophic results for failure [7]. Contrast with soft real-time systems, in which there are certain situations where changes in response timing might be permitted. Such systems introduce additional ecosystem variability, and so cannot be deployed in safety-critical applications. These requirements introduce specific constraints and behaviors of hard real-time vs soft real-time systems.

Hard Real-Time Applications:

- Response time requirements in the order of milliseconds or less
- Can result in a catastrophe if deadlines not met
- Peak-load performance must be predictable and should not violate the predefined deadlines
- Must remain synchronous with the state of the environment in all cases
- Temporal accuracy is often the concern here
- Often safety-critical

Soft Real-Time Applications:

- Response time requirements are higher
- Response times are less stringent
- A degraded operation in a rarely occurring peak load can be tolerated
- Possible slowed response time in high load situations is tolerable
- Non safety-critical [3]

Unlike hard real-time systems, soft-real time systems have greater tolerance for an occasional fault.

Aircraft flight control & antilock braking systems are good examples of hard real-time embedded systems in action. But there are other examples of systems where guaranteed timing performance isn't so critical.

Video streaming is one such application. Streaming is definitionally a real-time process–there really isn't much of a point to it unless packets are delivered relatively on time and in the proper order. But while a lapse in timing for a video application might result in frustration for the user, it will probably not result in loss of life. This is an example of a soft real-time system.

To summarize: in hard real-time systems timing performance matters in material & dramatic ways; soft real-time systems still operate in real-time, but with less critical performance needs.

Real-Time AI

Some AI systems operate in real-time. Examples of AIML systems which may be deployed as real-time applications include cybersecurity, finance, and autonomous vehicles. The latter is an example of AI deployed as a real-time system in a safety-critical context.

Real-Time Vs "Best Effort"

Real-time systems operate in contrast to *best effort* systems. Just like their name implies, these systems make a best effort to deliver data, but delivery is never guaranteed, and certainly not guaranteed within a specific deadline.

Best effort systems play a role in an overwhelming number of the most widely adopted technologies in use today. Internet Protocol (IP), operating systems like Unix, Linux, Mac and Windows, and even real-life systems such as the US Postal Service, are all examples of best effort systems.

Due to their **stochastic nature**, most AI systems are only suited for applications that fall into the best effort category. Real-time, embedded AIML systems require special engineering and **design assurance**.

References

1. Shaw, Alan C.. "Real-time systems and software." (2001).
2. Kant, Krishna (May 2010). Computer-Based Industrial Control. PHI Learning. ISBN 9788120339880.
3. Lee, Edward A.. "What Is Real Time Computing? A Personal View." IEEE Design & Test 35 (2018): 64-72.
4. "Real-Time Systems Overview and Examples." n.d. Intel. Accessed November 1, 2022. https://www.intel.com/content/www/us/en/robotics/real-time-systems.html.
5. Furht, B., Grostick, D., Gluch, D., Rabbat, G., Parker, J., McRoberts, M. (1991). Introduction to Real-Time Computing. In: Real-Time UNIX® Systems. The Kluwer International Series in Engineering and Computer Science, vol 121. Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-3978-0_1
6. "Real-Time Systems." n.d. Electrical and Computer Engineering. Accessed November 1, 2022. https://users.ece.cmu.edu/~koopman/des_s99/real_time/.
7. K. G. Shin and P. Ramanathan, "Real-time computing: a new discipline of computer science and engineering," in Proceedings of the IEEE, vol. 82, no. 1, pp. 6-24, Jan. 1994, doi: 10.1109/5.259423.