A Neglected/Ignored Research Topic in Real-Time Systems: **Timeliness in Mesosynchronous Real-Time Distributed Systems**

E. Douglas Jensen
The MITRE Corporation
http://www.real-time.org

Abstract

Traditional real-time computing concepts and techniques are focused on static, synchronous, relatively small-scale, mostly centralized, device-level subsystems. Many real-time systems, particularly distributed ones, are relatively large-scale, above the device level, and at least partially dynamic and asynchronous. We call such systems "mesosynchronous." For example, mesosynchronous systems often are found in military surveillance and force projection platforms, and in network-centric warfare (plus civilian domains). Hence the lives of both friends and foes depend on the timeliness properties of such systems being dependably acceptable according to application- and situation-specific criteria. The real-time research community has historically failed to perceive and appreciate this – admittedly difficult and domain-knowledge intensive – problem, especially for end-to-end timeliness in distributed mesosynchronous real-time systems.

1. Introduction

In the field of mesodynamics, the term *mesodynamic* refers to the middle ground between classical physics and quantum mechanics. By *mesosynchronous* real-time systems we mean those that are in the middle ground between

- Totally *synchronous* – in the sense of having only static, periodic, time-driven (i.e., TDMA-like) activities (or at least such activities are the only ones considered important)
- Totally *asynchronous* – in the sense of having only dynamic, aperiodic (not necessarily even sporadic), event-driven activities.

The derivation of "mesosynchronous" from "mesodynamic" reflects that: synchronous real-time computing, like classical physics, is comparatively well understood; while asynchronous real-time computing, like quantum mechanics, is still comparatively poorly understood, and seems to require a paradigm shift on the part of both the research and the practitioner communities [1].

It might be tempting to erroneously interpret "mesosynchronous" as meaning that a system is composed of separate traditional synchronous static hard real-time, and asynchronous dynamic, non-real-time parts. While mesosynchronous systems normally do have traditional synchronous hard real-time parts, the asynchronous parts are just as "real-time" as the synchronous ones are. And some parts are neither synchronous nor asynchronous – or are both. Properly speaking, a *real-time* activity is one that has a completion time constraint. Asynchronous activities may have deadlines, even hard deadlines that if missed result in operational failures – assurances about their timeliness are based on adherence to resource management policies, and are almost always unavoidably non-deterministic. More commonly, these activities have softer but more complex time constraints, and sequencing optimality criteria that are softer but more complex than simply always meeting all deadlines (e.g., minimize the expected completion time tardiness according to activity importance). That does not mean these activities are in any way less "important" or less mission-critical or even less safety-critical than the synchronous activities – indeed, quite the contrary.

2. Current and Future Mesosynchronous Systems

Mesosynchronous real-time systems, especially distributed ones, are (perhaps surprisingly to some people) very common.

For example, virtually all current and in-progress military platforms are mesosynchronous:

- surveillance and intelligence platforms – e.g., the AWACS, Joint Stars, Rivet Joint, and E-10A aircraft; the Global Hawk UAV; the SBIRS and Space-Based Radar satellites);
- force projection platforms – e.g., Aegis and DD(X) ships, the SSGN guided missile submarines, the Army's Future Combat System, UCAV's, bombers, fighters.

Network-centric warfare – which (among other things) replaces "smart" munitions with precision-guided munitions in a "smart" infrastructure for integrated sensing and control (e.g., the Affordable Moving Surface Target Engagement (AMSTE) system) – is mesosynchronous: e.g., the sensors are asynchronous; but the guidance updates to the missile are essentially synchronous (unlike guidance updates to interceptors against cruise missiles). Numerous examples exist in civilian application domains as well.

3. "Safety-Critical" Mesosynchronous Systems

Obviously, the purpose of military warfare systems is to save or destroy property and human lives – nothing could be more safety-critical.

Unfortunately, the term "safety critical" is reserved by convention for the tiny – albeit important – niche of small, static subsystems that can be developed and certified according to certain standards, notably RTCA's DO-178B (usually levels A or B).

Large scale mesosynchronous real-time systems often include some small synchronous subsystem(s) to which DO-178B processes can be applied. But these systems typically have millions or 10's of millions of lines of source code, and the overall system, and the applications in it, and their execution environment, are all inherently dynamic with many uncertainties. Neither current nor eventually foreseeable DO-178B-like development and certification processes are applicable to such systems. Normally, other, necessarily less formal and rigorous, approaches to assurance are employed.

More research is needed on development and certification processes – particularly for timeliness – for mesosynchronous real-time systems.

4. Priorities and Deadlines are Insufficient

In traditional real-time computing practice, time-criticality has been handled in one of two ways.

One way has been to attempt to map the application activities' inherent time constraints into an artifact called *priorities*, and to manage certain resources (notably, processor cycles) according to those priorities. The reason for this approach is that, although it may be feasible to reason about actual timeliness off-line, execution environments (OS's, JVM's, middleware) almost never provide support directly for time constraints and time constraint-based sequencing optimality criteria. They offer only priorities and priority-based sequencing. This has a number of serious disadvantages, including:

- In general, mapping time constraints into priorities is NP-hard. In practice, such mappings are semantically lossy, which makes it difficult to reason about timeliness, and hence makes it difficult to manage resources to dependably satisfy time-criticality requirements.
- Priority assignments are not modular – they require global knowledge of all other priority assignments (whereas time constraints, such as deadlines, do not). Such global knowledge is often difficult to obtain – for example, due to priority assignments being made by a multiplicity of designers and users in different organizational units, and having different limits on their security clearances.
- Relative importance is orthogonal to urgency, but priorities are usually the only mechanism available for expressing both, which inevitably results in overloaded semantics – again, making it difficult to reason about and dependably manage system behavior.

These disadvantages of priorities have proven to be severely costly in a variety of dimensions for non-trivial real-time systems – especially mesosynchronous ones.

The second popular way to handle time-criticality has been to attempt to over-provision resources (e.g., processor speed) so that timeliness objectives are met by brute force without having to explicitly take them into account when reasoning about the system and its mission, and hence when managing resources. This approach may be adequate in some cases. But in many other cases, it cannot be. The potential computational complexity of algorithms (e.g., track association) in many systems is essentially unlimited – no matter how fast the processors, some essential algorithms can consume that computational power, and more. In addition, computational hardware's size, weight, and power (not to mention cost) are all limiting factors in many system platforms.

The obvious solution to this problem is to provide system users and designers the direct abstraction of time-criticality, and to explicitly employ that abstraction for on-line reasoning about system and mission behavior, and hence for on-line management of resources to satisfy timeliness objectives.

The real-time research community (and a small fraction of the real-time practitioner community) does almost that – but for only a very limited subset of real-time systems: deadlines in periodic subsystems. Even there, the predominant model has been to map activity periods into priorities. Then resources are still managed by the execution environment using priorities.

Even if deadlines are used for reasoning about timeliness and managing resources, they suffer from weak expressiveness.

The extreme special case is in the context of conventional real-time computing (predominately research), where deadlines are only unit-valued binary expressions – a deadline is either met or missed. That context constitutes a very small part of the field of sequencing (usually meaning scheduling).

In general sequencing (e.g., scheduling) theory, which has a long history and vast body of scholarly literature compared to those of real-time computing, a deadline is a linear expression defined in terms of *lateness*=completion time–deadline. Although that formulation is more expressive than conventional real-time computing's binary special case, it is limited by being a linear expression.

That limitation imposes itself immediately in mesosynchronous real-time systems, which inevitably need richer time constraints (of which deadlines can be a special case as needed) and concomitant sequencing optimality criteria. Time/utility functions and utility accrual optimality criteria [1,2] are one approach that has been proven to be successful for an interesting class of mesosynchronous systems (e.g., [3]); no doubt there will be others.

The real-time computing research community could make an immensely valuable contribution to both the theory and the practice of real-time systems by broadening its attention to include sequencing in mesosynchronous systems.

5. Barriers to Research Progress

Several factors help explain why the real-time computing research community has not yet adequately begun to address the problems of timeliness in mesosynchronous real-time systems.

Most of the real-time computing research community fails to perceive much less appreciate the significance of mesosynchronous real-time systems and their need for concepts and technologies to ensure acceptable timeliness. One reason for this is that the community has only recently begun to emerge from more than a decade of concentration on scheduling hard (i.e., static periodic) real-time subsystems, conspicuously focused on rate-monotonic analysis. That concentration arose from the self-reinforcing cycle of research sponsor interest, and the intellectual and analytical tractability of the problem.

The easy tractability of the hard real-time scheduling problem (as with many others in many fields) is due in large part to it not requiring substantive knowledge about the real-time application domain – both the part of the domain for which the research was presumed to apply, and the part that was not perceived at all. Application domain independent research obviously has great potential advantages and great potential risks.

The disconnect between the real-time computing research community and the real-time application domain is a natural consequence of real-time computing primarily being a subset of embedded computing control systems. Historically, almost no academic research institution has had, or even has had access to, substantive systems with non-trivial real-time control requirements. That has recently begun to improve somewhat with the increase in research on real-time control of mobile autonomous platforms (e.g., vehicles), but even those represent a relatively small scale subset of real-time systems. Consulting has always been an alternative source of real world needs; but that has been impeded by the usual conflict between the customers' need for consultants with experience, and the consultants' need to gain experience. Some faculty (often with previous industrial employment) and students have been more successful than others in breaking that cycle. The most challenging – large scale, dynamic, mesosynchronous – real-time systems have always been, and continue to be, military ones. They present a special obstacle for academic researchers because most of the information about them is classified.

The result is that real-time computing has the biggest gap in all of computer science and engineering between the researchers and the real world. Researchers in compilers, operating systems, middleware, graphics, etc. all are accurately representative users in the field to which their research results are intended to apply. That is very rarely true for real-time computing researchers. Exceptions occur in the stereotypical hard real-time niche, and when enlightened sponsors or industrial enterprises arrange collaborative partnerships between researchers and significant real world projects. Fortunately, the trend is for an increasing number of such collaborations to form and succeed for both parties.

6. Conclusion

"The problem is never how to get new, innovative thoughts into your mind,
but how to get old ones out." [4]

References

[1] E. D. Jensen. Application qos-based time-critical automated resource management in battle management systems. Proc. IEEE Workshop on Object Oriented Real-Time Dependable Systems, October 2003.

[2] M. G. Gouda, Y.-W. Han, E. D. Jensen, W. D. Johnson, and R. Y. Kain. Distributed data processing technology, vol. iv, applications of ddp technology to bmd: Architectures and algorithms – chapter 3, radar scheduling: Section 1, the scheduling problem. Technical Report NTIS ADA047475, Honeywell Systems and Research Center, Minneapolis, MN, September 1977.

[3] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, P. Wallace, T. Wheeler, Y. Zhang, D. Wells, T. Lawrence, and P. Hurley. An adaptive, distributed airborne tracking system ("Process the right tracks at the right time"). In *Proceedings of the Seventh International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 1999),* April 1999.

[4] D. Hock, The birth of the chaordic age. Berrett-Koehler Publishers, ISBN 1576750744, January 2000.