Quasi-Synchronism: a step away from the traditional fault-tolerant real-time system models

Paulo Veríssimo
University of Lisboa *
FCUL

Carlos Almeida
Technical University of Lisboa
IST

Abstract

Distributed fault-tolerant real-time system models have exhibited a trend to polarize themselves in extreme positions.

In this paper, we assess the fitness of current models to represent the attributes underlying the distributed fault-tolerance of real-time systems. *Namely, we are concerned with the correctness issues arising from the temporal properties of interprocess communication: reliable and ordered group communication, replication management protocols, time services, etc.*

We are particularly concerned with best-effort *or* mission-critical *systems, where despite the* hard *need to fulfil timing guarantees, this cannot be ensured at all times in a given operational envelope, mandating that the system be highly dynamic and adaptive. In the paper, we suggest* quasi-synchronism *as a framework to address this kind of systems. We finalize by pointing out some contributions to materialize the model.*

1 Introduction

Fault-tolerant real-time system models, or in the context of this paper, *distributed* fault-tolerant real-time system models, have exhibited a trend to polarize themselves in extreme positions: time- or event-triggered, hard real-time or *not-at-all* real-time, and so forth.

What we care to address, in the context of this paper, is the fitness of current models to represent real-life distributed fault-tolerant systems, in the facet concerned with real-time properties. Namely, we are concerned with what underlies *distributed fault-tolerance of real-time systems*, that is to say, the correctness issues arising from the temporal properties of interprocess communication: reliable and ordered group communication, replication management protocols, time services, etc.

We will start by briefly analyzing the current models under the above-mentioned communication-oriented perspective, then we make our point about a possibly interesting model that we call *quasi-synchronism*, explaining what are the motivations for such a model. We finalize by pointing out some contributions to materialize the model.

2 The Models

Synchronism, as a fundamental system property, has separated the world of distributed systems in two main streams: asynchronous and synchronous. The latter being the obvious field of development of distributed real-time systems. A commonly accepted definition of synchronism can be described by the following system properties:

P 1 *Bounded and known processing speed*

P 2 *Bounded and known message delivery delays*

P 3 *Bounded and known local clock rate drift*

Which we would like to complement with the following additional properties, less commonly cited as contributing for a measure of system synchronism:

P 4 *Bounded and known load patterns*

P 5 *Bounded and known difference among local clocks*

*Department of Informatics, F.C.U.L. - Faculty of Sciences of the University of Lisboa, Bloco C5, Campo Grande, 1700 Lisboa - Portugal. Tel. +(351) 1 750 0084 (office). Fax +(351) 1 750 0103. E-mail: pjv@di.fc.ul.pt. The WWWeb: http://www.navigators.di.fc.ul.pt/. This work has been supported in part by the CEC, through Esprit Projects BROADCAST and GODC, and JNICT, through Programme PRAXIS-XXI. **This paper appeared in the IEEE TCOS Bulletin, Nr.4, Vol.7, December 1995.**

Property 4 defines the maximum load imposed on the system, without which no synchronism measure makes sense. Property 5 refers to the existence of synchronized clocks. With this, we are implying that there may exist real-time communication without a global clock. We will get back to this point later.

As for synchronous systems, the basis of real-time systems, two frequently opposing schools of thought have prevailed:

> **Time-triggered** system is one that reacts to significant external events at pre-specified instants.
>
> **Event-triggered** system is one that reacts to significant external events directly and immediately.

We have contributed[11] to show that these two models are not opposite, they simply serve different objectives, as to the kind of criticality, scale, dynamics, of the environment they are to be used on.

Furthermore, since we are focusing on communication and distributed processing, it is mandatory to finely characterize temporal properties of protocols. Which we have done by introducing two notions[19]: of the importance of the way in which protocols use clocks to secure temporal properties; of there existing a variable (and measurable) degree of synchronism of protocols.

The first notion is materialized by the following informal definitions:

> **Clock-driven** protocols rely on a *clock*, in the sense of a global time-base— that is, an absolute global time reference.
>
> **Timer-driven** protocols rely on local *timers*— i.e. relative time references.

The second notion is materialized by the following informal definition of degree of synchronism:

> **Steadiness** of a protocol is the greatest difference between message delivery delays, observed at one site, for all executions, all sites

For a formal treatment of these issues, please refer to [19].

3 Quasi-synchronism

The question is that real-life system requirements rarely fit completely into one of the definitions above. That is, the designer would like to be able to compound time- and event-triggered behavior, to use timer- and clock-driven protocols in his/her infrastructure, and so forth.

The reasons why can be explained by the growing use of distributed fault-tolerant real-time systems in areas outside life-critical applications such as nuclear plant control and avionics. The latter fit— by need of full control and testability of the design— in very simple, stable and static frameworks, whose predictable and repetitive behavior is very tractable by the so-called fully synchronous models, translated, in the context of this paper, to time-triggered, clock-driven behavior and, more importantly, very high coverage of the assumptions underlying the **Px** synchronism properties. When the application is a very critical one, having in its dependence human life or high cost resources, then one must make sure that all timing constraints are met.

There are, however, other real-time applications where, despite the need for dependability, meeting timing constraints is important but it is acceptable to eventually miss some of them, if the most important ones are achieved. Let us emphasize that this indulgence by the system user or designer often derives from the sheer impossibility of determining an operational envelope for those constraints: most of these applications are distributed, and have dynamic characteristics or run in dynamic environments, where it is difficult to evaluate a worst-case scenario, or the one obtained is so far from the normal case that it makes the use of a resource adequacy policy not cost-effective. Examples of this type of real-time application can be found in settings such as: factory plant control; command, control and communications systems; air traffic control.

In such situations, it is desirable to help the application programmer in adapting to this environment, by providing the adequate system support for interprocess communication and distributed processing. The problem has not deserved so far a complete treatment for dynamic and incompletely specified *real-time* settings. Indeed, most of the previously published work for dynamic and incompletely specified settings has treated the subject under asynchronous— that is *not-at-all-real-time*— assumptions and thus with no utility for our problem[4, 6, 14]. Other work[7] has addressed a similar issue, coming from the other extreme of the spectrum: how to restrict the asynchronism of a system— what the authors have called partial synchrony— to allow it to overcome a known result of impossibility of guaranteed consensus in fully-asynchronous systems[8]. However, this work being concerned with the eventuality of reaching consensus, did not address the issue of timeliness, or known bounds for delays. Finally, the problem of the kind of systems we are suggesting as our "target", has been

addressed at architectural level in previous works, such as Alpha[9] or DELTA-4[21] or DRTEE[3].

A bit of formalization

The first thing to emphasize is that these systems are **not** soft real-time systems in the strict sense of the word. The need to fulfill timing guarantees is *hard* in a given operational envelope. That is, the cost of not meeting those guarantees is commensurate to the benefit in absence of failure. However, unlike hard real-time systems[10], they are designed such that the probability of there being *timing faults* is non negligible, and as such imposes adequate timing-fault-tolerance measures, to avoid timing failures. Namely, detecting and recovering from timing faults. These systems have been commonly designated *best-effort* or *mission-critical*.

Synchronism in this framework can be formally defined, if the failure semantics can be formally defined (as in any other fault-tolerant system). We call it:

Quasi-synchronism- *a system is quasi-synchronous if:*

- *it can be defined by properties* **Px***;*
- *there is at least one bound where there is a known probability ($\neq 0$) that the bound assumption does not hold— this probability is called assumption uncoverage* [1]*;*
- *any property can be defined in terms of a series of pairs (bound, assumption uncoverage)*

By *quasi-synchronous* we mean that the bounds referred to in properties **Px** (process speed, message transmission delay, clock rate drift, etc.) exist, but some or all of them are too far from the normal case that in practice we are going to use other values (closer to the normal case). In this case, there is obviously a non-null probability that the values we pick are not correct.

The attentive reader is probably arguing at this moment that the applicability of our model depends on the capability of attaching realistic failure semantics to each property, and on an appropriate architecture capable of timing error processing, in order to achieve (timing-)fault-tolerance. We will address these issues in section 4.

For now, let us suggest that it is a realistic scenario to have a bi-modal distribution of a given bound, where there is a known and very high probability that the system's operational envelope lies within a smaller, "normal" bound, whereas it can assume, with correspondingly very low probability, a greater worst-case bound, in situations of overload, for example. We assumed a bi-modal distribution for simplicity of example: nothing prevents the system from being equated around a multi-modal distribution. Figure1 exemplifies best what we are talking about, with several bounds for maximum message delivery time, T_D, assumed in the distribution, with correspondingly smaller probabilities of being violated. In a classical system, only the rightmost bound with a acceptable residual probability of not holding, would be considered.

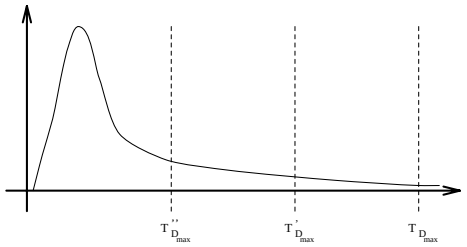

Figure 1: Distribution function for the message delivery time

Properties of a quasi-synchronous system

We now address the implications of quasi-synchronism in the **Px** properties. It is obvious that we can and should take some of them for granted even in a quasi-synchronous system. In this lot, we include properties 1 and 3.

Secondly, there is a property, 5, that depends on the existence and quality of clock synchronization. The issue has been well studied[17, 16], and can be self-contained, in a divide-and-conquer approach. Furthermore, securing property 5 is not mandatory to achieve real-time communication, as we discuss in [19], so we refer this discussion to the works cited above.

In consequence, we are left with properties 2 and 4. They are the crux of the problem we wish to discuss here. Since we are concerned with interprocess communication, we will further restrict the scope of property 4 to communication load, as opposed to processing load. Handling load has to do with scheduling resources, and though a global approach to schedulability would be desirable[5], this is a point of controversy[13]. It is perfectly obvious to us that this problem is only made worse in the kind of largely unpredictable environments we are bound to find in

[1] Coverage [12, 15] is a measure of the representativity of the situations to which a system is submitted during its validation compared to the actual situations it will be confronted with during its operational life.

the field of best-effort applications: systems composed of heterogeneous hosts over standard communication media, such as LANs or bridged LANs, or Internet. A divide-and-conquer strategy as was followed in [21], separating processing and communications scheduling, not being the perfect solution, has the merit of letting us focus on one problem at a time, and in this case, interprocess communication.

At this point, we also apply the divide-and-conquer approach to property 2, and split it into two properties:

P2. 1 Bounded and known access and transmission delay

P2. 2 Bounded and known number of transmission failures

That is, we separate what defines the bound of an individual network access, 1, from the number of network accesses needed to make the message reach through in the presence of failures, 2. This has the virtue of making property 4 be subsumed by property 2: access and transmission delay will depend on the load imposed on the communication system versus its capacity (throughput, latency), and we know how to model the network subsystem that way [20].

In conclusion, we simplify the interprocessing communication aspects of conceiving a quasi-synchronous distributed fault-tolerant system, to the task of handling property 2: achieving known and bounded message delivery delays.

4 Some contributions

The main implication of formalizing such a quasi-synchronism model is contributing to correctly specify fault-tolerant systems of the best-effort kind. That is, providing a framework for handling imprecise, coarse and highly variable timing assumptions. We have made some contributions towards this objective. Due to lack of space, we will briefly outline them and give pointers to the relevant papers.

Given that these systems are potentially nondeterministic, they tend to be treated in a homogeneous probabilistic framework, in sort of a soft real-time approach. With this kind of approach, no hard guarantees can be given. The quasi-synchronism model, on the other hand, allows the designer to define a set of operational envelopes, tracing a path of graceful degradation which the system goes through, in the measure where faults occur. Whenever timing assumptions are not met, it is essential to detect the fact and recover from it. In [1] and [2] we develop the innovative notion of *timing failure detector*, in contrast to the known crash failure detectors in asynchronous systems[6]. The timing failure detector is an oracle capable of telling which timing assumptions have been violated. This information is capable of triggering reconfiguration procedures, for example, migrating to another envelope.

Given the nature of these systems, they rely very much on event-triggered behavior, and timer-driven protocols. One obvious aim is to define a correctness setting for clock-driven and timer-driven protocols to coexist in a same system, and be proven correct. It should be understood that this was not allowed until now, for the sake of correctness of time-triggered systems[10]. In [19], we defined such generic correctness conditions based on fundamental properties of protocols such as steadiness defined in section 2. Not only have we showed they can coexist, but also that the normally coarse parameters and widely varying bounds expected of quasi-synchronous systems can yield ordering properties as useful as their highly-precise fully- or tightly-synchronous counterparts. The key of the argument lies in that a lot of real-time applications' requirements match the coarseness of system parameters such as the minimum and maximum propagation delays, or the minimum inter-event spacing.

In [18], we apply the above-mentioned reasoning to the ordering of events from time-stamps. We show that the classical attitude of having clocks as precise as possible, and with as low granularity as possible, does not necessary yield better results than coarser granularities. It all depends on what there is to be measured. Clearly, this result is well-suited to quasi-synchronous systems, since precision of clock synchronization depends, among other factors, on message delay variance.

In [20] we have addressed techniques to enforce properties 1 and 2 on real-time LANs, in the presence of overload and failures, that is, subsuming property 4, bounded and known load patterns, by the above-mentioned properties, bounded and known access and transmission delay and number of transmission failures.

In group communication, quasi-synchronism presents a suitable framework for protocols that usually have an execution time that is much faster than the worst-case (thus allowing the implementation of what is called early-delivery [1]), while still being able to work correctly in worst-case delays. Causal delivery, an important property in distributed system protocols, is generalized for real-time systems in [19]. There, it is also shown that coarse-granular protocols such as the ones one may find in quasi-synchronous systems, may

be as effective as their fine-granular counterparts, all depending on the system's parameters.

5 Conclusions

We addressed the shortcomings of current models to represent a class of real-life distributed fault-tolerant real-time systems, such as best-effort ones. After motivating our approach, we presented the model that we referred to as *quasi-synchronism*.

We pointed out some contributions to materialize the model, namely, contributing to correctly specify fault-tolerant systems of the best-effort kind. Specifically, in related works previously published we introduced: the notion of timing failure detector; generic correctness conditions for clock-driven and timer-driven protocols to coexist in a same system, and be proven correct; techniques to enforce bounded and known message delivery delay on real-time LANs, in the presence of overload and failures; real-time early-delivery and causal delivery protocols.

The work presented in this paper is still in progress. We intend to pursue the current development of a system architecture applying these concepts, and evaluate its merits for fault-tolerant best-effort systems.

References

[1] Carlos Almeida and Paulo Veríssimo. An adaptive real-time group communication protocol. In *Proceedings of the First IEEE Workshop on Factory Communication Systems*, Lausanne, Switzerland, October 1995.

[2] Carlos Almeida and Paulo Veríssimo. Timing failure detection and real-time group communication in *quasi-synchronous* systems. Technical Report RT/20-95, INESC, Lisboa, Portugal, November 1995. submitted for public.

[3] N. Audsley, K. Tindell, A. Burns, M. Richardson, and A. Wellinds. The drtee architecture for distributed hard real-time systems. In *Proceedings of the 10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Austria, September 1991. IFAC.

[4] Kenneth Birman, Andre Schiper, and Pat Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, 9(3), August 1991.

[5] A. Burns and A. Wellings. Real-time distributed computing. In *Proceedings of the 5th Workshop on Future Trends of Distributed Computing Systems*, pages 34–40, Cheju Island, Korea, August 1995.

[6] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for asynchronous systems (preliminary version). Technical report, Department of Computer Science, Cornell University, Ithaca, USA, July 1991.

[7] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the Association for Computing Machinery*, 32(2):374–382, April 1985.

[9] E. Douglas Jensen and J. Duane Northcutt. Alpha: A non-proprietary os for large, complex, distributed real-time systems. In *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, pages 35–41, Huntsville, Alabama, October 1990. IEEE.

[10] Hermann Kopetz, Andreas Damm, C. Koza, Marco Mulazzani, Wolfgang Schwabl, C. Senft, and R. Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The Mars Approach. *IEEE Micro*, pages 25–41, February 1989.

[11] Hermann Kopetz and Paulo Veríssimo. Real-time and Dependability Concepts. In S.J. Mullender, editor, *Distributed Systems, 2nd Edition*, ACM-Press, chapter 16, pages 411–446. Addison-Wesley, 1993.

[12] J. C. Laprie. Dependability: A Unifying Concept for Reliable Computing and Fault-Tolerance. In T. Anderson, editor, *Dependability of Resilient Computers*. BSP Professional Books, 1989.

[13] Gerard Le Lann. Timing failures and timeliness proofs in the case of distributed systems (draft). Technical report, September 1994. Presented at the Dagstuhl Seminar.

[14] Dalia Malki, Ken Birman, Aleta Ricciardi, and André Schiper. Uniform actions in asynchronous distributed systems. In *Proceedings of the 13th annual ACM symposium on the Principles of Distributed Computing (PODC)*, pages 274–284, Los Angeles, August 1994. also as TR 94-1447, Cornell University.

[15] David Powell. Failure mode assumptions and assumption coverage. In *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*, page 386. IEEE, 1992.

[16] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler. Fault-Tolerant Clock Synchronization in Distributed Systems. *IEEE, Computer*, pages 33–42, October 1990.

[17] Fred B. Schneider. Understanding protocols for byzantine clock synchronization. Technical report, Cornell University, Ithaca, New York, August 1987.

[18] P. Veríssimo. Ordering and Timeliness Requirements of Dependable Real-Time Programs. *Journal of Real-Time Systems, Kluwer Eds.*, 7(2):105–128, September 1994. Also as INESC AR/14-94.

[19] P. Veríssimo. Causal Delivery Protocols in Real-time Systems: a Generic Model. *Journal of Real-Time Systems*, to appear 1995.

[20] Paulo Veríssimo. Real-time Communication. In S.J. Mullender, editor, *Distributed Systems, 2nd Edition*, ACM-Press, chapter 17, pages 447–490. Addison-Wesley, 1993.

[21] Paulo Veríssimo, P. Barrett, P. Bond, A. Hilborne, L. Rodrigues, and D. Seaton. The Extra Performance Architecture (XPA). In D. Powell, editor, *Delta-4 - A Generic Architecture for Dependable Distributed Computing*, ESPRIT Research Reports, pages 211–266. Springer Verlag, November 1991.