

Real-Time Analysis of Round-based Distributed Algorithms

Alexander Kößler, Heinrich Moser, Ulrich Schmid
Embedded Computing Systems Group (E182/2)
Technische Universität Wien, 1040 Vienna, Austria
{koe,moser,s}@ecs.tuwien.ac.at

I. DISTRIBUTED COMPUTING VS. REAL-TIME SYSTEMS RESEARCH

Designing sound fault-tolerant distributed real-time systems requires a scientific basis, which allows to cope with three very different and partially conflicting major issues:

- (SD) *Spatial distribution*, i.e., multiple processors, typically coupled via some network(s), executing multiple processes that are working towards a common goal.
- (PF) *Partial failures* of system components, which may fail independently and without immediate recognition of each other.
- (RT) *Real-time requirements*, put on the response times of certain events by the environment. Additional concerns may be low power consumption, costs, etc.

The major consequence of (SD) and (PF) is *uncertainty* of the local processes about the global system state: One never knows exactly how far the execution of other processes proceeded and how long it takes for a message to arrive. Fault-tolerant distributed algorithms [1] have been invented to cope with this uncertainty.

Unfortunately, however, distributed algorithms research usually ignores real-time aspects: First, the wealth of research on (lock-step) *synchronous* systems assumes that all processes are perfectly synchronized by means of a common clock, and that all messages sent in step k are received by step $k + 1$. This is a very convenient restriction, as it rules out any uncertainty due to asynchrony and leaves only uncertainty due to failures; application-level modeling, (real-time) analysis and programming are hence easy. At the same time, however, it makes the system critically dependent on the continuous maintenance of synchronization: A synchronous system may even lose untimed safety properties like consistency of replicated data—not just timeliness properties—if synchronization is lost. Moreover, *implementing* synchrony (e.g. by means of a distributed clock synchronization algorithm [2], [3]) is costly and requires a priori bounds on end-to-end message delays.

Asynchronous distributed algorithms do not suffer from such problems. Unfortunately, however, modeling, (real-time) analysis, and programming are considerably more involved: (1) Most important distributed computing problems, like consensus, are impossible to solve in purely asynchronous systems in the presence of failures [4] (which makes *partially synchronous* (ParSync) systems like [5], [6], [7] attractive).

(2) The convenient time-triggered (periodic) invocation of processes, which is a prerequisite for all existing real-time scheduling approaches, is replaced by a message-driven invocation. (3) Timing aspects are usually abstracted away: Processes are modeled as state machines, which perform *zero-time* computing steps; time can only be modeled via the interval between computing steps here. Hence, the issues of queueing effects and scheduling do not arise at all.

As a consequence, the time complexity results obtained in distributed algorithms research are typically not particularly meaningful for real systems, and sometimes even too optimistic [8]. Bridging the gap between distributed algorithms and real-time systems research requires new approaches and analysis methods.

II. THE REAL-TIME DISTRIBUTED COMPUTING MODEL

We developed a *real-time distributed computing model* (RT model) [9], [10], [11], which replaces the zero-time steps of “classic” distributed computing models by non-zero-time *jobs*, shown in Figure 1, and hence allows to deal explicitly with queueing and scheduling. Most importantly, it allows to remove the classic assumption of *a priori* given end-to-end message delay bounds Δ , which are just considered as *model* parameters in classic distributed algorithms research. In reality, however, Δ not only depends on “raw” system parameters like computing step times μ and transmission delays δ , but also on the load (message transmissions, computations) created by a distributed algorithm \mathcal{A} , and the scheduling disciplines \mathcal{S} employed for processes and messages. Hence, $\Delta = F(\mathcal{A}, \delta, \mu, \mathcal{S})$ for some function F . On the other hand, the distributed algorithm \mathcal{A} itself may of course depend on Δ , just recall the round duration of a synchronous algorithm or the need for setting message time-outs. As a consequence, we usually have $\mathcal{A} = D(\Delta)$ for some function D . This creates a cyclic dependency of the algorithm and the end-to-end delays as shown in Figure 2, i.e., $\Delta = F(D(\Delta), \delta, \mu, \mathcal{S}) = F(D(\Delta), \delta, \mu, \mathcal{S})$.

To break the cyclic dependency, the “real” equation $\Delta = F(D(\Delta), \delta, \mu, \mathcal{S})$ must be derived and solved by means of a detailed real-time analysis. This analysis must incorporate the generated system load, the scheduling disciplines for processors and network, and of course the raw computing and transmission delays in order to derive a bound Δ that indeed holds when algorithm \mathcal{A} is executed in a given system. This

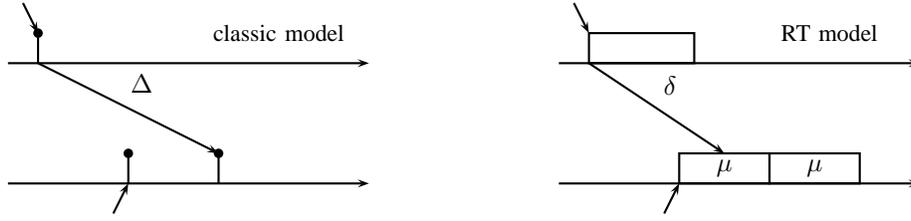


Figure 1. Comparison between the “classic” distributed computing model and “our” RT Model

paper is devoted to some preliminary results of our research on this problem.

III. ROUND DURATION OF A SYNCHRONOUS ALGORITHM: A SIMPLE EXAMPLE

We examined a simple “oral messages” Byzantine Generals algorithm [12], running on n processors in a completely synchronous lock-step round-based system. This algorithm works as follows: The *commander* (= one designated processor) broadcasts some initial value. Afterwards, each round basically consists of echoing all information heard so far, to ensure that, after $f + 1$ rounds, all (non-faulty) processors have enough information to decide on a common value, despite the fact that $f < n/3$ of these processors (including the commander) might be faulty and send out incorrect data.

Although this kind of “full information exchange” is a common pattern in fault-tolerant distributed algorithms, it causes the amount of data exchanged to increase exponentially with each round (reflected by an increasing number of messages). As it turns out [13], when using an optimal scheduling policy, the worst-case duration of a fault-free round for this algorithm is exactly $W = \max\{W^a, W^b, W^c\}$, with

$$\begin{aligned} W^a &= C + S \cdot \#_S + R \cdot \#_R, \\ W^b &= C + \delta^+ + R \cdot \#_R, \\ W^c &= C + S \cdot (\#_S - 1) + \delta^+ + R \cdot (n - r - 1), \end{aligned}$$

representing three possible critical paths. Herein, $C/S/R$ is the worst-case execution time of a computation/sending/receiving job, δ^+ is an upper bound on the message transmission delay, $\#_S/\#_R$ is the number of send/receive jobs in the current round, and r is the round number.

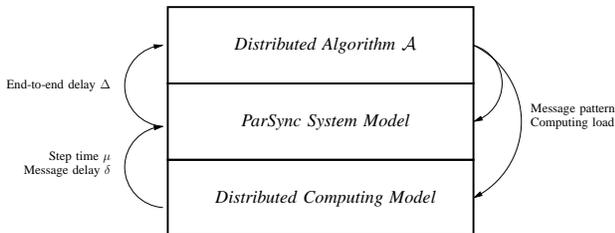


Figure 2. Dependency of the distributed computing model, the system model and a distributed algorithm.

IV. ROUND DURATION WITHOUT LOCK-STEP SYNCHRONY: AN OPEN PROBLEM

In the above “toy example”, all rounds start (periodically) in perfect synchrony. This convenient feature is lost in close-to-asynchronous ParSync algorithms, where some round $r + 1$ is started upon certain trigger events, like the arrival of the $n - f$ -th round r message from different processes. Since the latter’s round r starting times are not synchronized and the end-to-end delays may vary considerably, the round $r + 1$ starting times of different processors are typically even further apart. Keeping track of the resulting non-linear mutual dependencies requires a powerful mathematical method. Two promising approaches are illustrated below:

A. Max-Plus Algebra

When analyzing the time complexity of (non-fault-tolerant) asynchronous algorithms, one typically observes a certain structure of the resulting expressions, consisting of \max and $+$ operations only. To analyze such equations, there exists a well studied algebraic framework, namely, the Max-Plus algebra $\mathbb{R}_{max} = (\mathbb{R}_{max}, \oplus, \otimes, \epsilon, e)$ where $\epsilon := -\infty$, $e := 0$, and $\mathbb{R}_{max} := \mathbb{R} \cup \epsilon$. For elements $a, b \in \mathbb{R}_{max}$, the operators are defined as $a \oplus b := \max(a, b)$ and $a \otimes b := a + b$ [14]. To illustrate its use, consider the following simple distributed round synchronizer algorithm for the classic zero-step-time model: After initialization, every process (out of n) sends its round 1 message to every other process. If a process received all $n - 1$ round k messages, it switches to round $k + 1$ and broadcasts its round $k + 1$ message. The time series $x_i(k)$, $k \geq 1$, of every process i ’s round switching times, and derived quantities like the average round duration up to round k , can be calculated as follows: Given a transmission delay matrix $[\delta] \in \mathbb{R}^{n \times n}$ where $[\delta]_{ij} = \delta_{ij}$ is the delay of a message from process i to process j , we obtain the recursive formula $x_i(k) = \bigoplus_{j=1}^n x_j(k-1) + \delta_{ji}$. Using Max-Plus matrix multiplication with the delay matrix $[\delta]$ and introducing the round switching time vector $\vec{X}(k) \in \mathbb{R}_{max}^n := (x_1(k), x_2(k), \dots, x_n(k))^T$, this can be expanded to $\vec{X}(k) = (\delta^T)^{\otimes k} \otimes \vec{X}(0)$, with $\vec{X}(0)$ containing the initial starting times of the processes.

B. When Max-Plus is not Enough

Unfortunately, when making the transition to the RT model, where every incoming message triggers a non-zero-time job, queuing effects require an additional operation to be considered in time complexity expressions: the \min operation.

Note that the same is true, even in the classic model, when *fault-tolerant* asynchronous algorithms are considered. Dealing with the resulting equations requires the *Min-Max-Plus algebra* [15], where, in contrast to Max-Plus, not too many general results seem to be available yet.

We believe that combining the RT model and Min-Max-Plus algebra will result in a powerful framework for the analysis of partially synchronous fault-tolerant distributed algorithms. Needless to say, however, there is still a long way to go in order to solve the various research problems that arise in this context.

REFERENCES

- [1] N. Lynch, *Distributed Algorithms*. San Francisco, USA: Morgan Kaufman Publishers, Inc., 1996.
- [2] B. Simons, J. Lundelius-Welch, and N. Lynch, "An overview of clock synchronization," in *Fault-Tolerant Distributed Computing*, ser. LNCS 448, B. Simons and A. Spector, Eds. Springer Verlag, 1990, pp. 84–96. [Online]. Available: <http://faculty.cs.tamu.edu/welch/papers/lncs90.ps>
- [3] U. Schmid, Ed., *Special Issue on The Challenge of Global Time in Large-Scale Distributed Real-Time Systems*, ser. J. Real-Time Systems 12(1–3), 1997.
- [4] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.
- [5] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988.
- [6] J. Widder and U. Schmid, "The Theta-Model: Achieving synchrony without clocks," *Distributed Computing*, vol. 22, no. 1, pp. 29–47, Apr. 2009.
- [7] P. Robinson and U. Schmid, "The Asynchronous Bounded-Cycle Model," in *Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'08)*, ser. Lecture Notes in Computer Science, vol. 5340. Detroit, USA: Springer Verlag, Nov. 2008, pp. 246–262, (Best Paper Award). [Online]. Available: <http://www.vmars.tuwien.ac.at/php/pserver/extern/docdetail.php?DID=2398%&viewmode=paper&year=2008>
- [8] H. Moser and U. Schmid, "Optimal clock synchronization revisited: Upper and lower bounds in real-time systems," in *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, ser. LNCS 4305. Bordeaux & Saint-Emilion, France: Springer Verlag, Dec 2006, pp. 95–109. [Online]. Available: <http://www.vmars.tuwien.ac.at/php/pserver/extern/docdetail.php?DID=2068%&viewmode=paper&year=2006>
- [9] —, "Reconciling distributed computing models and real-time systems," in *Proceedings Work in Progress Session of the 27th IEEE Real-Time Systems Symposium (RTSS'06)*, Rio de Janeiro, Brazil, Dec 2006, pp. 73–76. [Online]. Available: <http://www.vmars.tuwien.ac.at/php/pserver/extern/docdetail.php?DID=2055%&viewmode=paper&year=2006>
- [10] H. Moser, "Towards a real-time distributed computing model," *Theoretical Computer Science*, vol. 410, no. 6–7, pp. 629–659, Feb 2009.
- [11] H. Moser and U. Schmid, "Optimal deterministic remote clock estimation in real-time systems," in *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, Luxor, Egypt, Dec. 2008, pp. 363–387.
- [12] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, July 1982.
- [13] H. Moser, "The byzantine generals' round duration," Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-2, 1040 Vienna, Austria, Research Report 9/2010, 2010.
- [14] B. Heidergott, G. J. Olsder, and J. von der Woude, *Max plus at work*. Princeton Univ. Press, 2006.
- [15] Y. Cheng, "A survey of the theory of min-max systems," in *Springer LNCS 3645 (Advances in Intelligent Computing ICIC'05, Part II)*, 2005, pp. 616–625.