Mechanical Verification of Clock Synchronization Algorithms *

D. Schwier and F. von Henke

schwier,vhenke@informatik.uni-ulm.de
Universität Ulm, Fakultät für Informatik,
D-89069 Ulm, Germany

Abstract. Clock synchronization algorithms play a crucial role in a variety of fault-tolerant distributed architectures. Although those algorithms are similar in their basic structure, the particular designs differ considerably, for instance in the way clock adjustments are computed. This paper develops a formal generic theory of clock synchronization algorithms which extracts the commonalities of specific algorithms and their correctness arguments; this generalizes previous work by Shankar and Miner by covering non-averaging adjustment functions, in addition to averaging algorithms. The generic theory is presented as a set of parameterized PVS theories, stating the general assumptions on parameters and demonstrating the verification of generic clock synchronization. The generic theory is then specialized to the class of algorithms using averaging functions, yielding a theory that corresponds to those of Shankar and Miner. As examples of the verification of concrete, published algorithms, the formal verification of an instance of an averaging algorithms (by Welch and Lynch [3]) and of a non-averaging algorithm (by Srikanth and Toueg [14]) is discussed.

1 Introduction

Clock synchronization is one of the central elements of distributed dependable real-time systems. Many mechanisms for realizing dependability properties in distributed real-time systems rely on the fact that the different processes or computing 'nodes' can be synchronized tightly enough for satisfying the real-time requirements of the system. A major concern is the ability to synchronize the local clocks of the different nodes in such a way that the readings of any two local clocks differ by no more than a small fixed amount. The synchronization algorithms for achieving this are required to compensate for the drift of physical clocks. Furthermore, they must be able to tolerate different kinds of failures, so that even if a limited number of processes fail the clocks of the remaining, properly functioning processes maintain the required synchrony.

Clock synchronization is thus a basic service that warrants careful analysis. The case for applying formal methods, including mechanized theorem proving,

* This work has been supported in part by ESPRIT LTR Project 20072 "Design for Validation (DeVa)" and ESPRIT Project 20716 "GUARDS'.

to this task has been made in the past (cf. [10]; see also [6] for a summary of previous work). Reasoning about fault-tolerant clock synchronization algorithms is inherently difficult because of the possibility of subtle interactions involving failed components. A proof with the assistance of a mechanized proof system thus offers a higher degree of assurance that the verification of a claimed property of a synchronization algorithm is indeed correct. However, since such proof efforts require substantial skill and effort, it appears to be very desirable to have available a reusable formal framework that assists in verifying the specific clock synchronization algorithms used in particular practically relevant contexts, such as [8,1,2].

Clocks are synchronized by the periodical application of an adjustment to the local clock value. The required bound between different clock values can be reached either by variation of the re-synchronization period length or by variation of the amount of adjustment. Clock synchronization algorithms can be classified as either averaging or non-averaging. *Averaging* algorithms synchronize clocks by variation of clock adjustments at regualr intervals. In contrast, *non-averaging* algorithms use a fixed clock adjustment and a varying period between clock adjustments.

Schneider [11] was the first to observe that correctness of averaging algorithms depends on common general assumptions about the applied convergence function. Subsequently, Shankar [13] verified Schneider's proof with the help of the EHDM system, a predecessor of the PVS verification system [7,6,5]. Miner [4] was able to relax some of the assumptions and extended the reasoning about recovery from transient faults. The theories developed by Shankar and Miner allow for a generic verification of algorithms that use an averaging function. The formalization of these clock synchronization algorithms takes a convergence function as a generic parameter; the underlying algorithm and its correctness argument remain fixed. In contrast, non-averaging algorithms like the one presented in [14] do not fit Miner's and Shankar's theories because they use a different algorithm and do not rely on a convergence function.

In this paper, we report on the formal analysis of a broader class of clock synchronization algorithms than those formally analyzed before. Our original starting point was a direct formalization of the non-averaging algorithm of Srikanth and Toueg [14] and of the averaging algorithm of Welch and Lynch [3]. Several commonalities and similarities between both types of algorithms became apparent, at the level of basic concepts (faulty/non-faulty processing nodes, interval clocks, rounds etc.) as well as at the abstract level of correctness arguments. This led to the development of a set of PVS theories, including a generic theory that generalizes the previous formalization of averaging clock synchronization algorithms by covering non-averaging algorithms, in addition to the more specialized theory of Shankar and Miner that deals with averaging algorithms. We regard as original contributions of this paper this more general theory and, as an instance of this theory, the formal, mechanically checked verification of the algorithm of Srikanth and Toueg [14], which, to our knowledge, is the first and so far only one for a non-averaging clock synchronization algorithm.

The remainder of this paper is organized as follows. The next section summarizes the basic concepts and the overall structure. Section 3 presents the generic theory and its specialization for the averaging case; Section 4 briefly describes the instantiation to concrete algorithms. The concluding section gives a summary and discusses ongoing work.

2 Overview

Clock synchronization algorithms operate on a cluster of nodes. Each node maintains a physical clock, typically a discrete counter that is incremented periodically; the *logical* clock of a node indicates its logical time, which is computed by adding an adjustment to its physical clock. The aim of clock synchronization is to keep the individual logical clocks of the nodes sufficiently well synchronized among each other (*agreement*). This is achieved by each node locally running an implementation of the clock synchronization algorithm and thereby periodically adjusting the clock. (We do not discuss accuracy, the second important property, in this paper.)

There are several ways to determine the clock adjustment. Averaging algorithms read the clock values of all or some of the nodes in the cluster to be synchronized. Then they calculate the adjustment from an average of all or some of the clock values read [11]. Non-averaging algorithms synchronize the clocks upon a special event, such as upon receipt of an explicit synchronization message on the network [14].

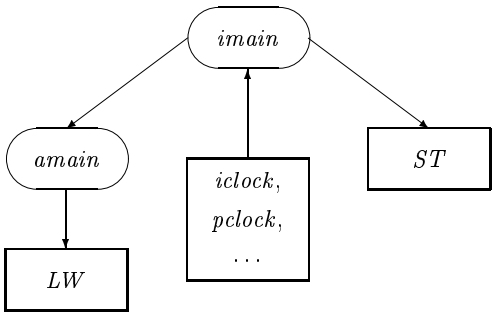

Fig. 1. Theory dependencies

The nodes communicate through an interconnection network. Each pair of nodes is able to exchange messages either directly or indirectly through remote nodes. Depending on the algorithm, different interconnection network architectures may be used. Furthermore, the network may or may not loose or fake messages. The general theories make no assumptions about the underlying network. This is left to the instances which formalize concrete algorithms.

The PVS formalization is structured into several theories. Each theory formalizes a certain concept and states the relevant theorems. Figure 1 shows the most important theories and their relationships. The auxiliary theories are not considered further in this paper; details can be found in the technical report that contains the full PVS specifications [12].

The PVS theories fall into three groups. The first group (*imain*, *iclock*, *pclock*, etc.) formalizes the basic concepts and theorems common to all synchronization algorithms. In theory *imain*, the central agreement property is stated and proved from fairly general assumptions. The proof is rather abstract and relies essentially on geometrical reasoning.

The second group of theories, consisting of PVS theory *amain* and auxiliary theories, formalizes averaging algorithms as a specialization of *imain*. Here, the central generic parameter is the convergence function, and a simpler set of assumptions is essentially sufficient to derive the agreement property. The assumptions about the convergence function are equivalent to those given by Miner [4]. The agreement theorem is obtained by instantiating the generic theory *imain*; this requires demonstrating that the generic assumptions of *imain* are satisfied by the actual parameters – which in turn is achieved by derivation from the assumptions on the convergence function.

The final theory formalizes the algorithm of Srikanth and Toueg [14] as an example of a non-averaging algorithm. Compared to the journal-level proof in ref. [14] which reasons about sets of nodes, our formulation of the lemmas is more appropriate for mechanized reasoning (e.g. by decision procedures) and enables PVS to find the required proofs largely automatically.

The validation of a concrete synchronization algorithm proceeds in two steps. First, the algorithm must be formalized; then an instance of the assumptions of one of the generic theories is derived from that formalization. For an averaging algorithm it is sufficient to provide an averaging function that satisfies the assumptions of theory *amain*; for a non-averaging algorithm one has to supply proofs for the assumptions of theory *imain*.

3 Generic Theories

Some PVS types and definitions are common to all theories. The type *nodeid* is the set of nodes that form the cluster. The predicate *faulty* formalizes the notion of faulty nodes. A node is faulty if it deviates from the algorithm or if its physical clock drifts too far from real time. Re-synchronization rounds are represented by the type *round*, which is equivalent to the type of natural numbers.

3.1 Clock Definitions

Each node maintains a physical clock which marks the passage of time. Real time is assumed to correspond to a Newtonian time frame. A physical clock typically is a discrete counter which is incremented periodically. $PC_i(t)$ is the physical clock value of node i at real time t. Clocks are modeled as functions

from *realtime* to *localtime*, where the type *localtime* is isomorphic to the natural numbers. The assumption *bounded drift* reflects the expected physical properties of the system. The rate at which a non-faulty clock can drift from real time is bounded by a small positive constant ρ:

$$\lfloor (t_2 - t_1)/(1+\rho) \rfloor \leq PC_i(t_2) - PC_i(t_1) \leq \lceil (t_2 - t_1)(1+\rho) \rceil$$

To simplify the presentation and analysis, the standard convention of interval clocks is adopted. Each of these interval clocks on a node is indexed by the number of rounds since the beginning of the run. At the r-th synchronization a node i starts a new interval clock, denoted by $IC_i^r(t)$. These interval clocks are derived from the physical clock by adding a round-specific adjustment.

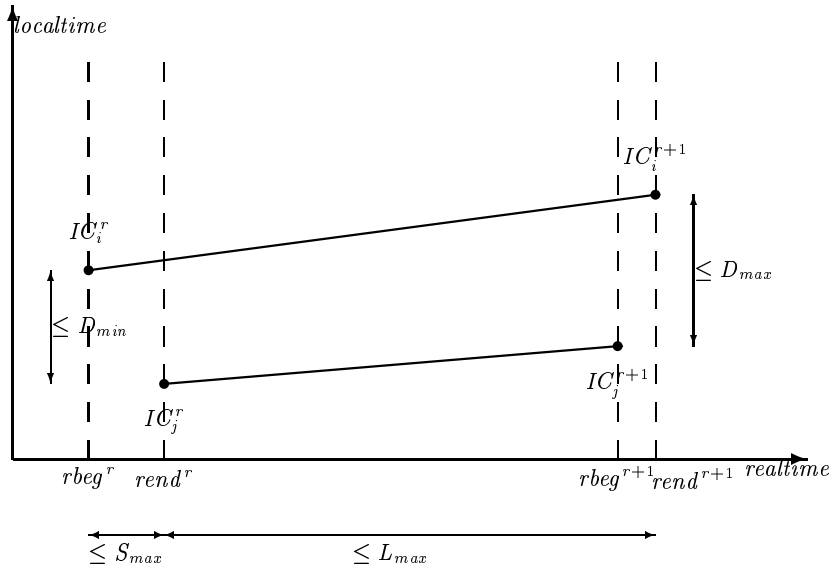

Fig. 2. *imain* constants

The symbol t_i^r denotes the real-time instant at which the interval clock IC_i^r starts. For every round there exists a first and a last node which start their new interval clocks. $rbeg^r$ and $rend^r$ are the minimum respectively the maximum of all t_i^r. At real time $rbeg^r$ the first node starts a new interval clock, while at real time $rend^r$ the last node does so. The discontinuously re-synchronizing interval clocks can be easily transformed into a single continuous logical clock. This can be achieved by spreading out each adjustment over the next re-synchronization

period. Several possible schemes for this are being discussed in ref. [14]; however, none of those has been formalized so far.

3.2 General Agreement

In theory *imain*, the main agreement theorem is stated and proved from the assumptions on the generic parameters. The parameters are various upper and lower bounds for time intervals. Figure 2 graphically illustrates the relationships among bounds and clock values.

imain$[\ldots,\ D_{min}, D_{max}, S_{max}, L_{max}, \ldots]$: **theory**
begin

assuming
% Less significant assumptions and parameters
% and imports of auxiliary theories are omitted.
...
rend_rbeg_upperbound : **assumption**
$rend^r - rbeg^r \leq S_{max}$
rend_rend_lowerbound : **assumption**
$rend^r \leq rend^{r+1}$
rend_rend_upperbound : **assumption**
$rend^{r+1} - rend^r \leq L_{max}$
startclock_bounded : **assumption**
$\left|IC_i^r(t_i^{r+1}) - IC_j^r(t_j^{r+1})\right| \leq D_{max}$
$\rightarrow\ \left|IC_i^{r+1}(t_i^{r+1}) - IC_j^{r+1}(t_j^{r+1})\right| \leq D_{min}$
init_assumption : **assumption**
$\left|IC_i^0(t_i^0) - IC_j^0(t_j^0)\right| \leq D_{min}$
const_assumption : **assumption**
$S_{max}(1+\rho) + D_{min} + (S_{max} + L_{max})\rho(2+\rho)/(1+\rho)\ \leq\ D_{max} - 2$
endassumption
...
agreement : **theorem**
$\left|IC_i^r(t) - IC_j^r(t)\right| \leq D_{max}$

end *imain*

The first assumption (*rend_rbeg_upper_bound*) constrains the length of the re-synchronization periods. Each correct node starts its r-th clock not later than S_{max} after the first one does so. The algorithm determines the point in time at which the next re-synchronization round begins. For *imain* an upper bound L_{max} for the period between re-synchronizations is sufficient (*rend_rbeg_upper_bound*). The inherent ordering of re-synchronization periods with respect to real time is reflected by assumption *rend_rend_lowerbound*. We assume that all logical clocks initially start at most a positive number D_{min} apart (*init_assumption*).

Assumption *start_clocks_bounded* is a precision enhancement condition: if all non-faulty clocks are at most D_{max} apart at the end of the previous round, the new clocks are started within the bound D_{min}.

The assumptions given in theory *imain* are sufficient to prove agreement of interval clocks. The proof is inductive on the round r. From the induction hypothesis that all non-faulty nodes were sufficiently synchronized during the previous round and the assumptions, one has to show synchronization for the current round. By the induction principle it follows that all non-faulty nodes are synchronized during all rounds. The induction step builds on a lemma (cf. Fig. 2) which states that if non-faulty clocks were at most D_{max} apart at the end of the previous round then they will also be at most D_{max} apart at the end of the current round. This lemma could be proved by the PVS system automatically without user interaction.

3.3 Averaging Algorithms

The generic theory of averaging algorithms is an instance of theory *imain*. The agreement theorem is inherited from *imain* by importing an aproporiate instance. An averaging algorithm uses a convergence function *cfn* to calculate the adjustment for the next round. *cfn* is a parameter of *amain*. The averaging algorithm assumes a mechanism to read clock values of remote nodes. Θ_i^r is the vector of all estimates at node i during round r. α and π are functions which calculate upper bounds for the convergence function.

Assumptions *start_clock_assumption* and *bound_on_read_error* reflect properties of the averaging algorithm. Whenever the local clock value is a multiple of the length R of a round, the adjustment is recalculated and applied. Even if the value of a remote clock can not be observed directly, the communication protocol allows a fairly accurate estimate (*bound_on_read_error*).

Two definitions simplify the notation of further assumptions. $read1(r, i, Y)$ constrains the difference between clock readings at a particular node. Clock readings on different nodes are constrained by $read2(r, i, j, X)$.

The next assumptions, describing properties of the convergence function *cfn*, are Miner's [4] conditions 3, 4 and 5, except for minor notational differences. *precision_enhancement* is a formalization of the concept that, after application of the convergence function, clocks should be close together. *accuracy_preservation* formalizes the notion that their should be a bound on the amount of correction applied in any synchronization interval. *translation_invariance* describes the insensitivity of the convergence function on absolute values of its arguments. More precisely, adding X to the result of the convergence function should be the same as adding X to each of the clock readings used in calculating the convergence function. The agreement theorem is obtained by importing theory *imain*.

4 Algorithm Instances

Two concrete algorithms serve as examples. The first one is the non-averaging algorithm described in [14]. The second is an averaging algorithm described

in [3]. For the instantiation of *imain* and *amain*, respectively, concrete values for the required functions and constants must be supplied. The assumptions of *imain* and *amain* must be derived from the formal description of the algorithms.

$amain[\ldots,\ cfn, \alpha(X), \pi(X, Y), \Delta, \beta, \ldots]$: **theory**
begin

assuming
% Less significant assumptions and parameters
% and imports of auxiliary theories are omitted.
...
start_clock_assumption : **assumption**
$(r+1)R = IC_i^r(t_i^{r+1})$
bound_on_read_error : **assumption**
$\left|IC_j^r(t_i^{r+1}) - \Theta_i^{r+1}(j)\right| \leq \Delta$
$read1(r, i, Y) \ := \ \forall\, l, m \cdot |\Theta_i^r(l) - \Theta_i^r(m)| \leq Y$
$read2(r, i, j, X) \ := \ \forall\, l \cdot \left|\Theta_i^r(l) - \Theta_j^r(l)\right| \leq X$
precision_enhancement : **assumption**
$\exists\, X, Y \cdot read1(r, i, Y) \wedge read1(r, j, Y) \wedge read2(r, i, j, X)$
$\rightarrow \left|cfn(i, \Theta_i^r) - cfn(j, \Theta_j^r)\right| \leq \pi(X, Y)$
accuracy_preservation : **assumption**
$\exists\, Y \cdot read1(r, i, Y)$
$\rightarrow \left|cfn(i, \Theta_i^r) - IC_j^r(t_j^r)\right| \leq \alpha(Y)$
translation_invariance : **assumption**
$cfn(i, \Theta_r^i + X)) = cfn(i, \Theta_i^r) + X$
initialization : **assumption**
$\left|t_i^0 - t_j^0\right| \leq \beta$
endassuming
...
importing $imain[\ldots, \alpha(\beta + 2\delta)/(2\rho R + \beta), \alpha(\beta + 2\delta),$
$2\rho R + \beta, (1 + \rho)R + \beta, \ldots]$
end *amain*

4.1 Srikanth-Toueg Algorithm

The clock synchronization algorithm of Srikanth and Toueg [14] relies on authenticated synchronization messages and broadcasting. During every round, a synchronization message (*round* r) is processed. Whenever the local clock is a multiple of R, a node prepares a (*round* r) message, signs it, and broadcasts it on the net. Concurrently, each node accepts and processes messages originating from remote nodes. Whenever a node has received at least $f + 1$ (*round* r) messages, it starts a new clock and relays all $f + 1$ messages to all other nodes.

Because both activities run concurrently, a node may start a new clock before it becomes ready. Roughly speaking, this algorithm leads to a synchronization with the fastest logical clock.

This algorithm tolerates at most f faulty nodes if the number n of all nodes is larger than $2f + 1$. A non-faulty node which has received more than $f + 1$ messages has received at least one synchronization message originated from a non-faulty node; even if there where f faulty messages on the net. Relaying all received messages ensures that all non-faulty nodes start there new interval clocks. For details of the algorithm see [14] and the formalization in [12].

The agreement property is obtained by instantiating theory *imain* with the following concrete values: $\lfloor t_{del}(1+\rho) \rfloor$ for D_{min}, $\lfloor S_{max}(1+\rho) \rfloor + D_{min} + \lfloor S_{max} + L_{max} \rfloor (\rho(2+\rho)/(1+\rho)) + 2$ for D_{max}, t_{del} for S_{max} and $(P - \alpha)(1+\rho)$ for L_{max}. Because of space limitation, only the proof of *rend_rbeg_upperbound* is sketched. The first non-faulty node starts its clock at $rbeg^r$. At this point in time the node has received $f + 1$ messages. Since it relays all these messages, every correct node receives at least $f + 1$ messages and starts its clock by time $rbeg^r + t_{del}$, where t_{del} is the maximal network delay. Thus $rend^r - rbeg^r \leq t_{del}$. The other assumptions are proved in a similar manner.

4.2 Averaging Instance: Fault-Tolerant Midpoint

Agreement for the Lynch/Welch algorithm is attained by instanting theory *amain* with concrete parameters: $(\lfloor \theta_{(f+1)} + \theta_{n-f} \rfloor)/2$ for *cfn*, the identity function for $\alpha(X)$ and $Y/2 + X$ for $\pi(X, Y)$. The proofs of the instantiated assumptions *precision_enhancement*, *accuracy_preservation* and *translation_invariance* where essentially follow Miner's [4]. Details can be found in [12].

5 Conclusion

This paper has describes PVS theories that give a unified framework for proving agreement for averaging and non-averaging clock synchronization algorithms. Agreement is proved in the general theory from a set of generic assumptions. The general theory has been specialized to a theory for averaging algorithms which inherits the agreement theorem from the general theory. Two concrete algorithms [3,14] were given a of the main theories. A full report [12], including the complete PVS specifications, will be made available online.

There are several directions in which the current work can be expanded. The present formalization does not consider initialization and re-integration of clocks. Also, further specific clock synchronization algorithms and convergence functions should be examined as to how well they fit into the general framework presented here. We are currently working on fitting the clock synchronization of the Time-Triggered Architecture (TTA) [1,2] into the framework; this requires a slight modification of the theories because of the different way in which the clock values of other nodes are gathered. A further direction of current work is an examination of how the clock synchronization theories can be combined with

Rushby's general approach to verifying fault-tolerant time-triggered systems [9], which is based on, and assumes, synchronized clocks.

Acknowledgements. We would like to thank Ercüment Canver and Holger Pfeifer for valuable discussions; Ercüment Canver also contributed directly to this work in an early stage.

References

1. H. Kopetz and G. Gruensteidl. Ttp – a time triggered protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, January 1994.
2. Hermann Kopetz. The time-triggered approach to real-time system design. In B. Randell, J.-C. Laprie, H. Kopetz, and B. Littlewood, editors, *Predictably Dependable Computing Systems*. Springer, 1995.
3. J. Lundelius Welch and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Inf. and Comp.*, 77(1):1–36, 1988.
4. P. S. Miner. Verification of fault-tolerant clock synchronization systems. NASA Technical Paper 3349, NASA Langley Research Center, Nov. 1993.
5. S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.
6. S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. *IEEE Trans. on Software Engineering*, 21(2):107–125, February 1995.
7. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.
8. D. Powell. A Generic Upgradable Architecture for Real-Time Dependable Systems. In *IEEE Intern. Workshop on Embedded Fault-Tolerant Systems, Boston*. May 1998.
9. J. Rushby. Systematic formal verification for fault-tolerant time-triggered algorithms. In M. Dal Cin, C. Meadows, and W. H. Sanders, editors, *Dependable Computing for Critical Applications—6*, pages 203–222. IEEE Computer Society, March 1997.
10. J. M. Rushby and F. von Henke. Formal verification of algorithms for critical systems. *IEEE Trans. on Software Engineering*, 19(1):13–23, Jan. 1993.
11. F. B. Schneider. Understanding protocols for byzantine clock synchronization. Technical Report 87-859, Cornell University, Aug. 1987.
12. D. Schwier and F. von Henke. Mechanical verification of clock synchronization algorithms. Ulmer Informatik Berichte, Universität Ulm, 1998 (forthcoming).
13. N. Shankar. Mechanical verification of a schematic byzantine clock synchronization algorithm. Technical Report CR-4386, NASA, 1991.
14. T. K. Srikanth and S. Toueg. Optimal clock synchronization. *Journ. of the ACM*, 34(3):626–645, July 1987.