Adaptive Internal Clock Synchronization

Zbigniew Jerzak*, Robert Fach, Christof Fetzer
Dresden University of Technology
Systems Engineering Group
D-01062 Dresden, Germany
{Zbigniew.Jerzak, Robert.Fach, Christof.Fetzer}@inf.tu-dresden.de

Abstract

Existing clock synchronization algorithms assume a bounded clock reading error. This, in turn, results in an inflexible design that typically requires node crashes whenever the given bound might be violated. We propose a novel, adaptive internal clock synchronization algorithm which allows to compute the deviation between the clocks during runtime. The computed deviation can be propagated to the application layer to allow it to adapt its behavior according to the current clock deviation. The contributions of this paper are: (1) a new specification of a relaxed clock synchronization problem, and (2) a new clock synchronization algorithm with a novel approach to dealing with crash failures.

Keywords: internal clock synchronization, real-time, timed asynchronous systems, distributed systems

1 Introduction and Related Work

A distributed system is composed of a set of processes communicating via messages. The processes do not have access to a global clock. A large number of distributed algorithms [17] and systems [15, 13] rely, however, on synchronized clocks that provide the same view of time for all processes. An algorithm that ensures a common view of time is called a clock synchronization algorithm.

Synchronizing clocks is difficult, and in some systems, even impossible. First of all, various message transmission delays prevent a perfect and instantaneous view of the clocks of other processes in the system. Moreover, even if all clocks were to be initially perfectly synchronized, i.e., the values of all clocks would be identical, they would, with time, drift away from each other due to the clocks' oscillators having slightly different frequencies than the ones specified by the clock manufacturer. Oscillator frequencies not only differ from nominal values, but they also fluctuate with time due to the external conditions such as temperature or aging. Finally, message omissions and process crashes contribute to the difficulty of the clock synchronization problem.

A common view of time in a distributed system is achieved by synchronizing clocks with a clock synchronization algorithm. Clocks can be synchronized with respect to an external time source [3, 2] (external clock synchronization), with respect to each other [23, 7] (internal clock synchronization) or both [9, 21]. Moreover, clock synchronization algorithm can be either deterministic [6, 16], i.e., assuming an upper bound on message transmission delays and thus maximum difference between any two clock values (precision), or probabilistic [3], where no such assumption is being made.

Internal clock synchronization bounds the deviation of the software clocks of the correct processes in a system [23, 24] by an a priori known value Δ, called *precision*. Such a bound cannot be guaranteed in asynchronous distributed systems which are characterized by having no a priori known maximum message transmission delay. In this paper, we propose a new **adaptive internal clock synchronization** specification based on the fail-aware approach to construction of hard real-time applications [12]. The goal of the adaptive clock synchronization is (1) to minimize the deviation between the clocks of the correct processes, and (2) to provide a local bound that indicates how well the local clock is currently synchronized. This maximum clock synchronization error E_p is computed locally by each non-crashed process p during runtime. It states an upper bound on the current deviation between the value of the clock of the local process and the value of some *abstract* (and unknown) clock in the system: if two clocks are E_p and E_q apart from this abstract clock, they know that they are at most $E_p + E_q$ apart from each other. In a first approximation, adaptive clock synchronization is similar to external clock synchronization where the abstract clock takes the role of the external clock. However, the value of the abstract

*work supported by the Polish Ministry of Science and Higher Education grant number N N516 375034.

1060-9857/08 $25.00 © 2008 IEEE
DOI 10.1109/SRDS.2008.32

clock is a function (which is in this paper the midpoint) of all clock values.

To explain the usage of adaptive clock synchronization, consider that a process p is assigned a time slot $[S, T]$ in which it can exclusively access some resource R, e.g., a broadcast network. The two time stamps S and T are defined with respect to the abstract clock. We must make sure that process p is not accessing R outside its time slot, i.e., before S or after T. We can use the dynamically calculated error E_p to ensure this requirement even if p's clock is not synchronized within an a priori known constant: p must not use R before p's local time $S + E_p$ and not after local time $T - E_p$.

Our specification of adaptive clock synchronization does not guarantee that a system will obtain an a priori specified synchronization precision. However, in most systems one will achieve a sufficiently small clock deviation for most of the time. The advantage of the new specification is that, unlike other approaches like [23, 7, 18], we do not require a bound on the maximum number of crashed processes nor do we need to assume a bound on the maximum initial clock deviation. Also, unlike other deterministic approaches [7, 6], we do not require an upper bound on message transmission delays. In particular, adaptive clock synchronization can be implemented in systems with omission, performance and crash failures like the Timed Asynchronous Distributed System Model [5].

Crash failures of process can increase the deviation of clocks. In this paper, we also propose a new method for extending the validity of the remote clock readings beyond a single synchronization interval. The new method, called clock readings extrapolation, allows for masking of transient process performance failures and message omissions. This permits us to deal with crash failures of processes in a graceful manner.

2 System Model

We implement the adaptive clock synchronization algorithm in a system based on the Timed Asynchronous Distributed System Model (TADSM) [5]. The TADSM is sufficiently weak to represent dependable distributed systems and sufficiently strong to facilitate the building of adaptive real-time systems.

2.1 Processes and Hardware Clocks

We assume that a system consists of a set $\mathbf{N} = \{p_1, \ldots, p_n\}$ of timed processes, i.e., there exists a time interval σ within which every correct process is supposed to answer a request sent to it. However, there are no guarantees that a request is indeed being processed in the given time period. We assume processes have crash-stop/performance failure semantics. A process might not respond within σ seconds because it is either slow (overloaded) or it has crashed. We transform value (Byzantine) failures caused by wrong executions into crash failures using the Software Encoded Processing [22] approach developed in our group.

Interprocess communication is performed using an unreliable transport protocol with omission/performance failure semantics. Messages sent between processes are supposed to be delivered within δ seconds. However, they might be delayed, delivered out of order or might get dropped. Specifically, there exists no upper bound on the frequency of communication and process failures. Every process $p \in \mathbf{N}$ has access to a local hardware clock, where the value $H_p(t)$ denotes the content of the local hardware clock of process p at real-time t. Hardware clocks have a drift rate that is bounded by an a priori known constant ρ:

$$\forall_{p \in \mathbf{N}} \forall t : \quad |\rho_p(t)| \leq \rho \tag{1}$$

where $\rho_p(t)$ is the drift rate of the hardware clock of a process p at real-time t. For commercial, off-the-shelf (COTS) components the value of ρ usually stays within the range of $[1ppm; 100ppm]$ [14]. A hardware clock is said to be correct if it stays within the linear envelope of the real-time:

$$(t - s)(1 - \rho) \leq H_p(t) - H_p(s) \leq (t - s)(1 + \rho) \tag{2}$$

where $[s, t]$ is an arbitrary real-time interval. We assume hardware clocks expose crash-stop failure semantics. Specifically, we use an approach presented in [10] to convert non-maskable, arbitrary hardware clock failures into crash-stop failures.

2.2 Clock Synchronization

The clock synchronization algorithm proposed in this paper is an instance of a generic clock synchronization algorithm used throughout the literature [23, 18, 1]. In order to synchronize the clocks of all processes, the clock synchronization algorithm is executed periodically by every process $p \in \mathbf{N}$. A single period of the clock synchronization algorithm is called a *synchronization interval* and has a fixed length of P seconds. During a single synchronization interval, the clock synchronization algorithm performs two basic steps: (1) it estimates the values of remote clocks using remote clock reading (see Section 2.4), and (2) it calculates and applies a clock adjustment value to the local hardware clock so that the values of the clocks converge.

During each synchronization interval, process p executing the clock synchronization algorithm (see Listing 1) reads the clocks of all processes. Every remote clock reading is stored along with accompanying remote clock reading error in the `clk[]` and `err[]` variables, respectively. Subsequently, process p calculates a local adjustment value

```
ClockVal A_p; // current adjustment
ClockVal T; // end of current round

void init() {
    (A_p,T) = initialAdjustment();
    // every P starting at T
    schedule(synchronizer, P, T);
}

void synchronizer() {
    // N - number of processes
    ClockVal clk[N];
    ClockVal err[N];
    // remote clock reading
    readClocks(clk, err);
    // adjustment for the next round
    A_p = adjust(A_p, T, clk, err);
    // set T to next round
    T = T + P;
}
```

Listing 1. Gerneric internal clock synchronization algorithm

A_p which is to be applied to the local hardware clock during the next synchronization interval. The behavior of the hardware clock is thus determined by the discrete $adjust\,()$ function. We follow the definition in [4, 7] and define the $adjust\,()$ function as:

$$\text{adjust}(A_p, T, \text{clk}, \text{err}) = A_p + \text{cfn}(\text{clk}, \text{err}) - T \quad (3)$$

where $\text{cfn}(\text{clk}, \text{err})$ represents a convergence function. We define D_p to be the change of the adjustment value A_p, i.e., $D_p = \text{cfn}(\text{clk}, \text{err}) - T$. A convergence function is applied to the data provided by the remote clock reading. It calculates a local view of the point in time to which all processes should converge. We discuss convergence functions in more detail in Sections 3 and 4.

2.3 Software Clocks

The discrete local clock adjustment value A_p which is calculated by process p based on the results of the remote clock readings is not applied directly to the local hardware clock. Instead, software clocks are used. Formally, we define a software clock of a given process p as a function transforming local hardware clock time $H_p(t)$ of this process to a software clock time $S_p(t)$:

$$S_p(t) = H_p(t) + a_p(t) \quad (4)$$

where $a_p(t)$ is a function of real-time, based on the discrete local clock adjustment value A_p calculated by the clock synchronization algorithm of the process p.

In the following, we define $a_p(t)$ as a continuous function of real time and derive it by spreading the local discrete adjustment value D_p across the whole next synchronization interval of length P. This is achieved by changing the speed Φ_p of the local software clock S_p of the process p during the next synchronization interval:

$$\Phi_p = \begin{cases} \min\left(\frac{D_p}{P}, k\rho\right) & \text{if } D_p > 0 \\ \max\left(\frac{D_p}{P}, -k\rho\right) & \text{otherwise} \end{cases} \quad (5)$$

where k is a small constant chosen to satisfy:

$$k\rho \geq 2\rho + \rho^2 \quad (6)$$

$$k\rho \ll 1 \quad (7)$$

as it has been defined in [19]. The presence of the $k\rho$ term in Equation 5 might not allow us to reach the target local adjustment value A_p for the software clock at the end of the next synchronization interval. However, it bounds the maximum drift of a software clock and prevents trivial implementations, i.e., setting all clocks to zero.

The calculated software clock speed Φ_p can be used to express the software clock adjustment value $a_p(t)$ (and thus the value of the software clock $S_p(t)$ of the process p) for the next synchronization interval:

$$a_p(t) = a_p + (H_p(t) - H_p) \cdot \Phi_p \quad (8)$$

where a_p is an adjustment value at the end of current synchronization interval, $H_p(t)$ is the current value of the p's hardware clock and H_p is the value of the hardware clock of the process p at the beginning of the synchronization interval within which $a_p(t)$ is valid:

$$H_p \leq H_p(t) < H_p + P \quad (9)$$

The above defined $a_p(t)$ allows us to construct a continuous and monotonic software clock $S_p(t)$ (see Equation 4).

2.4 Remote Clock Reading

In order to calculate the local discrete adjustment value A_p (see Equation 3), process p has to obtain the values of the remote clocks with accompanying remote clock reading errors – `clk[]` and `err[]`, respectively. This is achieved by the remote clock reading method. The remote clock reading is based on the approach presented in [11, 14] – due to space constraints, we need to refer the reader to the aforementioned papers for more details.

3 Problem Statement

Ideally, we would like that all clocks are perfectly synchronized, i.e., at any point in real-time t, the value of the

local software clock $S_p(t)$ of the process p is the same as the value of any other process q:

$$\forall_{p,q\in\mathbf{N}}: \quad S_p(t) = S_q(t) \tag{10}$$

However, because of the varying drift rates of hardware clocks and the inability to obtain a perfect view of the local hardware clocks of other processes (see Sections 2.1 and 2.4), it is not possible to assume that software clocks will maintain the above property. The requirement of **adaptive clock synchronization** is (a) to minimize the difference between the software clocks of the correct processes and (b) to bound it by the local maximum clock synchronization errors:

$$\forall_{p,q\in\mathbf{N}}\forall_t: \quad |S_p(t) - S_q(t)| \leq E_p(t) + E_q(t) \tag{11}$$

The maximum local clock synchronization error $E_p(t)$ is computed locally, during runtime by every correct process $p \in \mathbf{N}$. The goal of the clock synchronization algorithm, i.e., the minimization of the value of $E_p(t)$. We define a simple metric that defines the quality of an adaptive clock synchronization algorithm A:

$$Q(A) = \max_{\forall_t,\forall_{p\in\mathbf{N}}:p\ \neg\mathrm{crashed}(t)} \{E_p(t)\} \tag{12}$$

This metric permits us to compare two adaptive clock synchronization algorithms and in particular, trivial algorithms that set $E_p(t) = \infty$ are assigned a bad quality.

An adaptive clock synchronization algorithm is based on a convergence function which determines the local discrete clock adjustment value A_p and in this way adjusts the value of the local software clock $S_p(t)$ (see Equations 3 and 4). A convergence function $\mathrm{cfn}()$ computes the midpoint of the interval I_p^i containing the values of the remote clock readings (`clk[]`) performed by the process p during the ith synchronization interval. We define a midpoint $\mathrm{mid}\,([x,y])$ of an interval $[x,y]$ as:

$$\mathrm{mid}\,([x,y]) = \frac{x+y}{2} \tag{13}$$

The convergence function $\mathrm{cfn}()$ is thus given by:

$$\mathrm{cfn}\,() = \mathrm{mid}\left(\left[L_p^i, U_p^i\right]\right) \tag{14}$$

where L_p^i and U_p^i are lower and upper bound on the remote clock reading values (`clk[]` – see Listing 1) obtained by the process p in the ith synchronization interval.

We define the maximum local clock synchronization error $E_p(T)$ as the function of the software clock time T to be the difference between the midpoint of all correct clocks in the system and the value of the local software clock $S_p(t)$. Formally, we state:

$$E_p(T) = |C_p(T,p) - \mathrm{cfn}()| + \mathcal{E}_p(T) \tag{15}$$

where $\mathcal{E}_p(T)$ is the error on the calculation of the midpoint value $\mathrm{cfn}()$. Intuitively, if we were able to read all remote clocks at the same instance of time with a reading error equal zero, the calculated midpoint value $\mathrm{cfn}()$ would be precise. However, due to the clock drift rates and the clock reading errors, the calculated midpoint value needs to be bounded by an error $\mathcal{E}_p(T)$. We define the midpoint calculation error as:

$$\mathcal{E}_p(T) = \max\left\{\mathrm{cfn}() - \overleftarrow{M}_p(T), \overrightarrow{M}_p(T) - \mathrm{cfn}()\right\} \tag{16}$$

where $\overleftarrow{M}_p(T)$ and $\overrightarrow{M}_p(T)$ are the lower and upper bounds on the possible midpoint values obtained based on the remote clock readings ($C_r(T,p)$) and remote clock reading errors ($\varepsilon_r(T,p)$):

$$\overleftarrow{M}_p(T) = \mathrm{mid}\left(\left[\overleftarrow{L}_p(T), \overleftarrow{U}_p(T)\right]\right) \tag{17}$$

$$\overrightarrow{M}_p(T) = \mathrm{mid}\left(\left[\overrightarrow{L}_p(T), \overrightarrow{U}_p(T)\right]\right) \tag{18}$$

The values of the interval boundaries $\overleftarrow{L}_p(T)$, $\overleftarrow{U}_p(T)$, $\overrightarrow{L}_p(T)$ and $\overrightarrow{U}_p(T)$ are defined separately for each convergence function in Section 4.

The maximum local clock synchronization error is a measure of synchronization of a single process with respect to the midpoint of the correct processes in the system. In the introduction, we view this midpoint as an abstract clock to which all processes try to synchronize to; very much like in external clock synchronization.

4 Convergence Functions

In this section, we present two convergence functions $\mathrm{cfn}_{V0}()$ and $\mathrm{cfn}_{V1}()$ derived from the Differential Fault-Tolerant Midpoint Function [7]. The convergence functions are represented by the definitions of the lower and upper bounds on the remote clock reading values (L^i, U^i – see Equation 14).

The first convergence function, $\mathrm{cfn}_{V0}()$, is build upon the one presented in [23], except we do not remove any of the remote clock readings from the set upon which the $\mathrm{cfn}_{V0}()$ is evaluated (which is only required when considering arbitrary failures). We show that the maximum local synchronization error (see Equation 15) and the extrapolation of the clock readings (see Section 5) allow us to preserve the fault-tolerant properties without bounding the maximum number of failed processes. Moreover, we use the fail-aware membership protocol [8] to eventually exclude crashed processes. The $\mathrm{cfn}_{V0}()$ is therefore calculated by a process p as:

$$L_{p,V0}^i = \min_{r\in N}\{C_r^i(T,p)\} \tag{19}$$

$$U_{p,V0}^i = \max_{r\in N}\{C_r^i(T,p)\} \tag{20}$$

The bounds ($\overleftarrow{M}_p(T)$ and $\overrightarrow{M}_p(T)$) on the possible midpoint values are defined by (see Equations 17 and 18):

$$\begin{aligned}
\overleftarrow{L}_p(T) &= \min_{r\in\mathbf{N}}\{C_r(T,p)-\varepsilon_r(T,p)\}\\
\overleftarrow{U}_p(T) &= \max_{r\in\mathbf{N}}\{C_r(T,p)-\varepsilon_r(T,p)\}\\
\overrightarrow{L}_p(T) &= \min_{r\in\mathbf{N}}\{C_r(T,p)+\varepsilon_r(T,p)\}\\
\overrightarrow{U}_p(T) &= \max_{r\in\mathbf{N}}\{C_r(T,p)+\varepsilon_r(T,p)\}
\end{aligned}$$

The second convergence function $cfn_{V1}()$ is based on the approach presented in [4]. One goal of [4] was to deal with process crash failures during resynchronization. The idea was to extend the interval (which contains the correct value of a remote clock) computed by remote clock reading method such that one can handle a bounded number of omission and crash failures: even if a process cannot read all clocks, the interval extension makes sure that the intervals of non-crashed processes are sufficiently close to each other and hence, clocks stay in sync. We discuss $cfn_{V1}()$ to compare it with our new method to deal with crash and omission failures.

We need to modify the convergence function of [4] because (1) we do not need to handle arbitrary failures, and (2) we permit arbitrary high clock reading failures. The interval extension is defined with the help of term $X_r^i(T,p)$:

$$X_r^i(T,p)=\begin{cases}\Lambda-\varepsilon_r^i(T,p) & \text{if } \varepsilon_r^i(T,p)\le\Lambda\\ \varepsilon_r^i(T,p) & \text{otherwise}\end{cases} \tag{21}$$

where Λ is the "target" for the remote clock reading error, i.e., the system tries to keep the clock reading error below this value.

The lower and upper bound on the remote clock reading values calculated by a process p are thus given by:

$$L_{p,V1}^i = \min_{r\in\mathbf{N}}\left\{C_r^i(T,p)-X_r^i(T,p)\right\} \tag{22}$$

$$U_{p,V1}^i = \max_{r\in\mathbf{N}}\left\{C_r^i(T,p)+X_r^i(T,p)\right\} \tag{23}$$

The bounds ($\overleftarrow{M}_p(T)$ and $\overrightarrow{M}_p(T)$) on the possible midpoint values are defined by (see Equations 17 and 18):

$$\begin{aligned}
\overleftarrow{L}_p(T) &= \min_{r\in\mathbf{N}}\{C_r(T,p)-\max\{\Lambda,\varepsilon_r(T,p)\}\}\\
\overleftarrow{U}_p(T) &= \max_{r\in\mathbf{N}}\{C_r(T,p)-\max\{\Lambda,\varepsilon_r(T,p)\}\}\\
\overrightarrow{L}_p(T) &= \min_{r\in\mathbf{N}}\{C_r(T,p)+\max\{\Lambda,\varepsilon_r(T,p)\}\}\\
\overrightarrow{U}_p(T) &= \max_{r\in\mathbf{N}}\{C_r(T,p)+\max\{\Lambda,\varepsilon_r(T,p)\}\}
\end{aligned}$$

5 Clock Readings Extrapolation

The remote clock readings for convergence functions $\mathrm{cfn}_{V0}()$ and $\mathrm{cfn}_{V1}()$ are only valid during the current synchronization interval. During round $i+1$, all clocks have to be reread. A problem we need to address is the crash of a process p: the membership protocol might not remove p before the next clock synchronization round. Hence, the remaining processes will not be able to read p's clock. Even worse, some process q might be able to read p's clock just before p crashed while another process r might not be able to read p's clock anymore. Hence, processes p and q cannot just ignore clocks that they cannot read because otherwise they would disagree of the midpoint of the clocks.

Therefore, we propose a new way to extrapolate remote clock reading methods to be able to deal with crashed clocks (not yet removed by the membership protocol) and also excessive reading errors because of performance and omission failures. We extrapolate remote clock readings from round i to round $i+1$ and we use this extrapolation to improve the values obtained via remote clock reading.

Specifically, let us assume that process p has access to the clock reading value $C_q^I(T,p)$ as well as the reading error value $\varepsilon_q^I(T,p)$ of the process q from the previous, ith synchronization round (see Figure 1). In the current synchronization round $i+1$, following the round i, process p reads the clock of the process q using the remote clock reading (see Section 2.4). Knowing that the synchronization round $i+1$ is executed P seconds after synchronization round i we denote the so obtained clock reading value and clock reading error as: $C_q(T+P,p)$ and $\varepsilon_q(T+P,p)$, respectively. For the same synchronization round $i+1$ we extrapolate the values from the round i. The extrapolation result for clock reading value and clock reading error is thus given by $C_q^E(T+P,p)$ and $\varepsilon_q^E(T+P,p)$, respectively, and is the function of the reading from the previous round i:

$$C_q^E(T+P,p) = C_q^I(T,p)+P \tag{24}$$

$$\varepsilon_q^E(T+P,p) = \varepsilon_q^I(T,p)+(k+2)\rho P \tag{25}$$

In the above equation we shift the clock reading by P (the length of the synchronization round) seconds and simultaneously we increase the clock reading error so as to accommodate the maximum distance ($(k+2)\rho P$) a software clock and a hardware clock can drift apart during the duration of the synchronization round.

The last step of the extrapolation method is the calculation of the intersection of the two readings: the remote clock reading: $C_q(T+P,p)$ and $\varepsilon_q(T+P,p)$ with the extrapolated reading from previous, ith, synchronization round: $C_q^E(T+P,p)$ and $\varepsilon_q^E(T+P,p)$. The intuition behind the intersection is that since both methods guarantee correct bounds on the clock reading values and the error of the clock readings, we can assume that their intersection satisfies this property as well. More formally, we define the clock reading intersection $C_q^I(T+P,p)$ and error of the clock reading intersection $\varepsilon_q^I(T+P,p)$ for the reading of

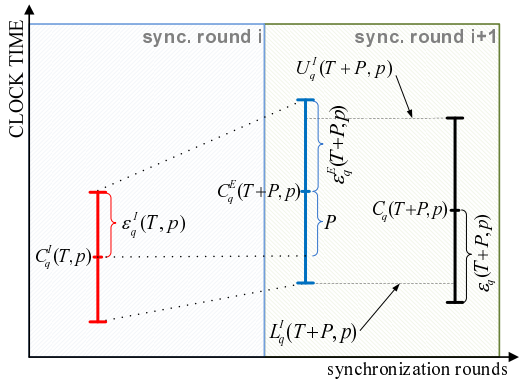

Figure 1. Extrapolation of the clock readings

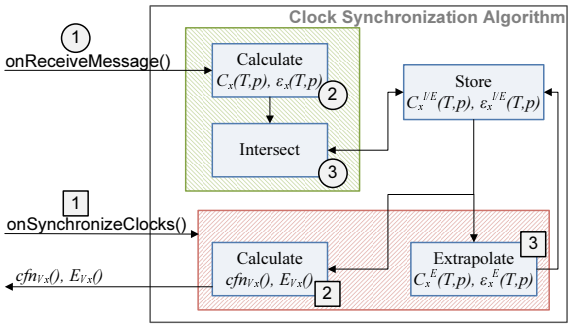

Figure 2. Clock synchronization algorithm implementation

the clock of the process q by the process p as:

$$C_q^I(T+P,p) = \text{mid}\left(\left[L_q^I(T+P,p),\ U_q^I(T+P,p)\right]\right) \quad (26)$$

$$\varepsilon_q^I(T+P,p) = \frac{U_q^I(T+P,p) - L_q^I(T+P,p)}{2} \quad (27)$$

where $L_q^I(T+P,p)$ and $U_q^I(T+P,p)$ are the lower and upper bound on the intersection interval for the synchronization round $i+1$:

$$L_q^I(T+P,p) = \max\{\ C_q(T+P,p) - \varepsilon_q(T+P,p),\ C_q^E(T+P,p) - \varepsilon_q^E(T+P,p)\} \quad (28)$$

$$U_q^I(T+P,p) = \min\{\ C_q(T+P,p) + \varepsilon_q(T+P,p),\ C_q^E(T+P,p) + \varepsilon_q^E(T+P,p)\} \quad (29)$$

The intersection of the extrapolated and calculated clock readings allows us to improve the clock reading errors, as it is only possible to narrow the error values $\varepsilon_q(T+P,p)$ obtained by remote clock reading method.

Extending the above calculations to readings of all processes by the process p in the synchronization round $i+1$, we obtain a new set of clock readings and accompanying reading errors. This, in turn, allows us to use the extrapolated values as the input to the convergence functions.

We also evaluated convergence functions $\text{cfn}_{V0}()$ and $\text{cfn}_{V1}()$ (see Section 4) together with the extrapolated and intersected clock readings and errors. We refer to this applications of the two convergence functions as $\text{cfn}_{V0,1}()$ and $\text{cfn}_{V1,1}()$.

6 Algorithm Overview

Figure 2 shows the internal clock synchronization algorithm as executed by the process p. Process p receives messages with time stamps via an upcall `onReceiveMessage()`. Upon reception of a message, process p calculates the remote clock reading value $C_x(T,p)$ and remote clock reading error $\varepsilon_x(T,p)$ for that message (x denotes the process sending the message). Subsequently, process p replaces the intersection/extrapolation value pair $C_x^{I/E}(T,p)$ and $\varepsilon_x^{I/E}(T,p)$ with its intersection with the just calculated remote clock reading value pair $C_x(T,p)$ and $\varepsilon_x(T,p)$. The calculation of the convergence function value $cfn_{Vx}()$ and the accompanying local maximum clock synchronization error $E_p(T)$ is triggered by an asynchronous call at time T to the `onSynchronizeClocks()` method, which itself is a marker for the beginning of the new synchronization round. The `onSynchronizeClocks()` method uses the stored intersection/extrapolation value pair $C_x^{I/E}(T,p)$ and $\varepsilon_x^{I/E}(T,p)$ to calculate and return the $cfn_{Vx}()$ and $E_{Vx}()$ values. Subsequently, the algorithm invalidates the intersection/extrapolation value pair $C_x^{I/E}(T,p)$ and $\varepsilon_x^{I/E}(T,p)$ by overwriting it with its own extrapolation for the new synchronization round.

7 Evaluation

For evaluation purposes, we have developed and simulated a hard real-time distributed system using TDMA schedule to drive the communication over a half-duplex 10Mbit/s Ethernet shared bus. The simulation has been performed with the aid of the OMNeT++ discrete event simulation system [20] with the INET Framework.

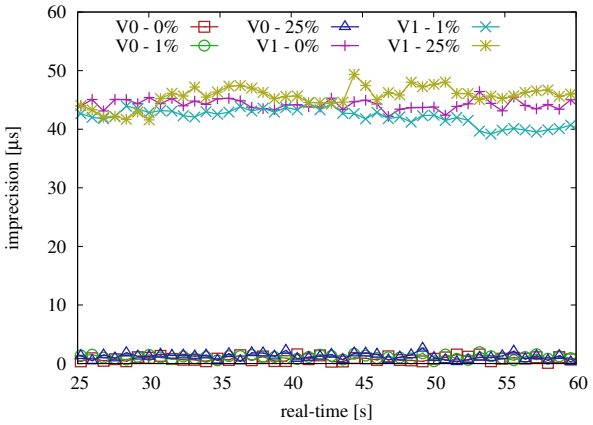

Figure 3. Clock synchronization imprecision (no extrapolation)

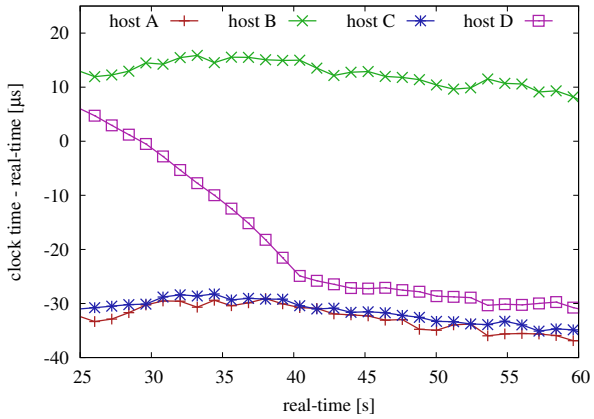

Figure 4. Clock time minus real-time for $\mathrm{cfn}_{V1}()$

Our choice of the simulation is motivated by the fact that certain measurements, e.g., the global observer view, can only be performed in the simulation environment. We have simulated four hosts, placed in a row with 10 meters distance between successive hosts. The signal propagation speed was set to 200 km/s. In order to mimic the behavior of real-life hardware clocks we have conducted a number of experiments on the PlanetLab to obtain the drift rate characteristics of the hardware clocks of modern computers [14]. These observations allow us to implement the hardware clock in the simulation environment in such a way as to mimic the behavior of the changing drift rates of real-world hardware clocks. For all experiments, we have assumed the upper bound on the simulated hardware clock drift rate to be equal to 80 ppm.

Figure 3 shows the global observer view of the imprecision of the clock synchronization algorithm when using $\mathrm{cfn}_{V0}()$ and $\mathrm{cfn}_{V1}()$ convergence functions. The global observer view of the imprecision is measured as the maximum distance between the global views of the software clocks of all processes at the given real-time instance. The presented graph shows the imprecision for the three varying omission probabilities: 0%, 1% and 25%.

The difference in the imprecision between the $\mathrm{cfn}_{V0}()$ and $\mathrm{cfn}_{V1}()$ convergence function can be explained with the help of Figure 4. We can observe that the Λ extension results in a channel of about $50\mu s$ between the software clocks. This behavior is caused by the local process forming both the lower and upper bounds: L_{V1} and U_{V1} (see Equations 22 and 23). The local process can form both bounds as the reading error of its local software clock is by definition equal to zero and hence, the Λ extension value is, in the case, greater than of the other processes. Only when the clocks drift apart sufficiently far such that another clock determines one of the bounds, the clocks can converge closer again.

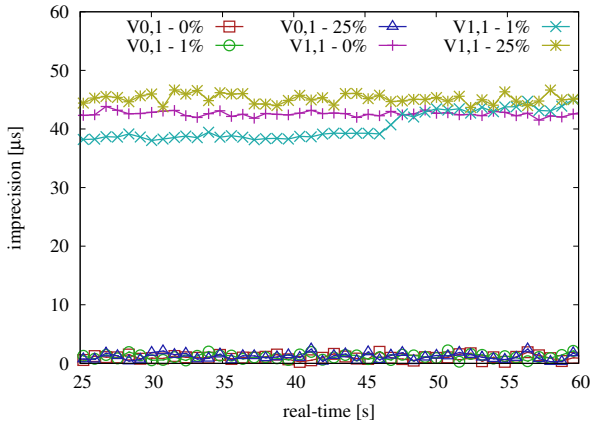

Figure 5. Clock synchronization imprecision (with extrapolation and intersection)

Figure 5 shows the analogous global view of the imprecision of the clock synchronization algorithm for $\mathrm{cfn}_{V0,1}()$ and $\mathrm{cfn}_{V1,1}()$ convergence functions. We can observe that the extrapolation and intersection techniques do not significantly influence the imprecision when compared with the original convergence functions (see Figure 3). The explanation for that fact is demonstrated on Figure 6 which shows the mean improvement in % of the intersected readings over the non-intersected readings. The mean is calculated from the beginning of the run up to the current position.

The improvement is calculated as one minus the ratio of the length of interval $\left[C_p^I - \varepsilon_p^I,\ C_p^I + \varepsilon_p^I\right]$ and the smaller of the two intervals: $[C_p - \varepsilon_p,\ C_p + \varepsilon_p]$ or $\left[C_p^E - \varepsilon_p^E,\ C_p^E + \varepsilon_p^E\right]$ (see Section 5). The difference in improvement for the host D and host A in the 1% omissions case can be explained by the fact that host D has been defined with lower transmission delays (on average three times) than host A and thus it is unlikely that the extrapo-

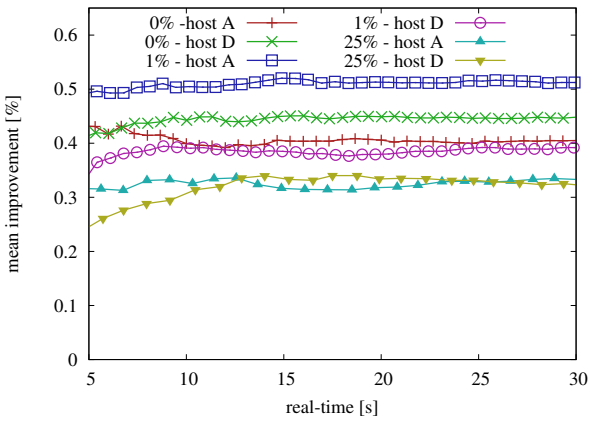

Figure 6. Mean improvement using the intersection approach for $\mathrm{cfn}_{V0,1}()$

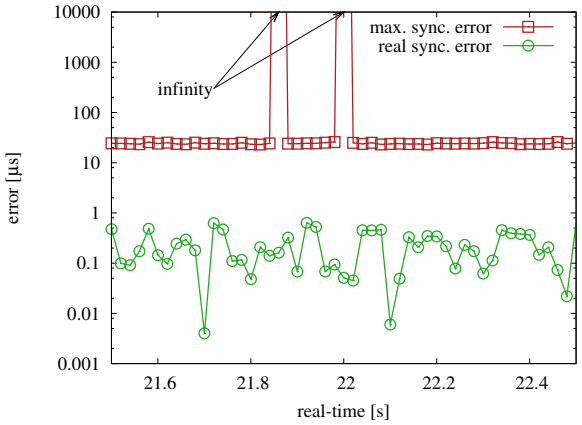

Figure 7. Local maximum clock synchronization error vs the real clock synchronization error (no extrapolation) – $\mathrm{cfn}_{V0}()$

lated reading error is smaller than the reading error from the current round. Analogously, for the 25% omissions cases it is less likely that the current reading error is lower than the extrapolated one, as it is often not possible to obtain a reading at all. Therefore, in such cases, the calculated improvement is lower.

The motivation for the clock readings extrapolation is presented in Figure 7. It shows a comparison between the maximum local synchronization error $E_p(T)$ (see Equation 15) calculated by one of the processes and the real clock synchronization error as perceived by the global observer. The real clock synchronization error describes the global observer view of the distance between the local clock and the midpoint of the values of software clocks of all processes. Figure 7 shows the local synchronization error as calculated by one of the processes using the $\mathrm{cfn}_{V0}()$ convergence function with a 25% probability of omissions.

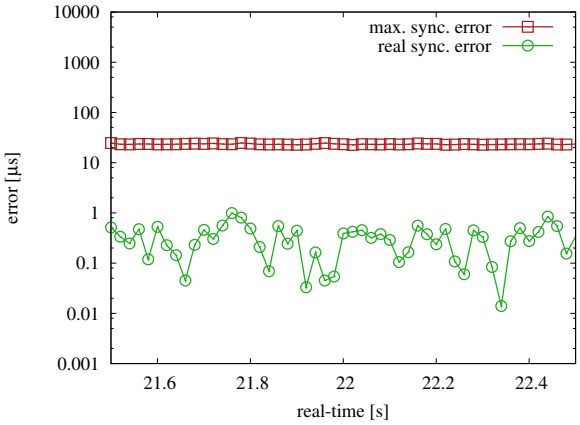

Figure 8. Local maximum clock synchronization error vs the real clock synchronization error – $\mathrm{cfn}_{V0,1}()$

We can observe that the real synchronization error is violated whenever a process is not able to read at least one of the remote clocks. In such cases the values of $\overleftarrow{M}_p(T)$ and $\overrightarrow{M}_p(T)$ (see Equations 17 and 18) are assigned the logical values of infinity, because a given process is not able to obtain the values of the remote clock readings and accompanying remote clock reading errors. The resulting maximum synchronisation error (see Equation 15) is thus also equal to infinity.

Figure 8 shows how the clock readings extrapolation copes with the above problem. Extrapolation of the old readings allows us to mask the transient remote clock reading failures and provides us with a correct bound on the maximum local clock synchronization error (see Equation 15).

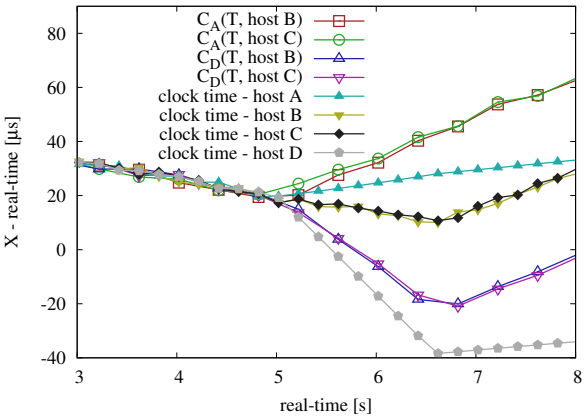

Figure 9. Clock time minus real-time and remote clock readings for two processes suffering omission failures – $\mathrm{cfn}_{V0,1}()$

We have also simulated a case when two processes (host A and host D) in the system suffer a permanent asymmetric omission failure in that they do not receive any messages from other processes, however are able to send messages themselves. We have simulated the omissions to be triggered when 5 real-time seconds elapse. Figure 9 shows the difference between the software clock time and real-time of all four processes. Moreover, we can see the local view of the software clocks of the processes suffering the omission failure as perceived by the correct processes (host B and host C). We can observe that both hosts B and C calculate the correct midpoint with respect to their readings of the hosts A ($C_A(T, B/C)$) and D ($C_D(T, B/C)$). Moreover, we can see that the remote clock readings divert from the correct software clock values (clock time – host A/D ; this is caused by to the aging of the helper messages used by the remote clock reading method). Note that the membership protocol will eventually remove A and B because they suffer permanent omission failures.

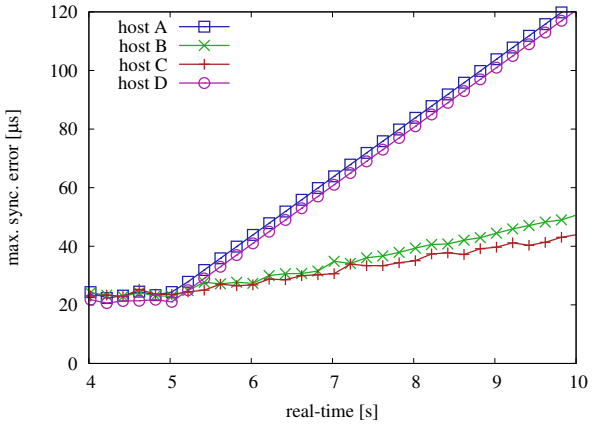

Figure 10. Maximum local synchronization error for two processes suffering omission failures – $\mathrm{cfn}_{V0,1}()$

For the above run, we have also plotted the maximum local synchronization error – see Figure 10. We can observe that the maximum local synchronization error of processes A and D is quickly rising – as they are unable to read any remote clock and have to extrapolate the old clock readings. Simultaneously, the maximum clock synchronization error of the correct processes rises only as they drift away from the correct midpoint due to the influence of the clocks suffering the asymmetric omission failures.

Figure 11 shows the run from the Figure 10 with the membership protocol [8] enabled. We can observe that the hosts suffering omission failures (A and D) are excluded by the membership service when they drift too far apart from the correct clocks. Simultaneously, we can observe that the correct clocks (host C and D) maintain low maximum local

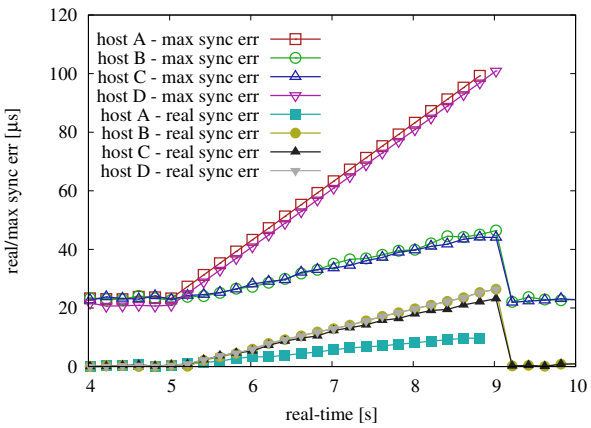

Figure 11. Maximum local synchronization error vs real synchronization error for two processes suffering omission failures – $\mathrm{cfn}_{V0,1}()$

synchronization error, which tightly reflects the changes in the real synchronization error. After the membership protocol excludes the faulty processes both local and real maximum synchronization errors return to the stable state from before the omissions occurrence.

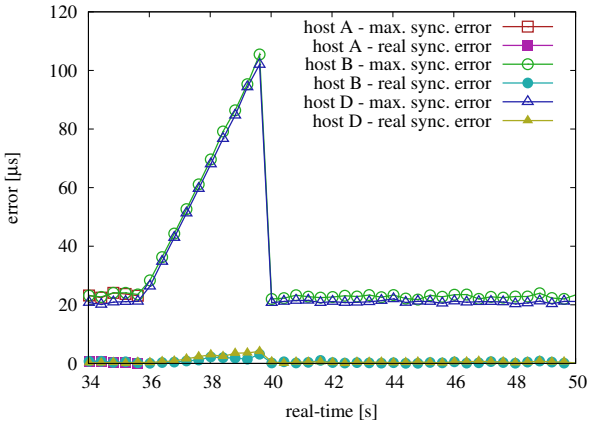

Figure 12. Local maximum clock synchronization error vs the real clock synchronization error (host A crashes) – $\mathrm{cfn}_{V0,1}()$

To test the correctness of the membership protocol we have plotted the maximum local synchronization error and the real synchronization error for $\mathrm{cfn}_{V0,1}()$ when one of the processes (host A) crashes (see Figure 12). We can observe that the calculated local maximum synchronization error increases after the crash of the host A until the crashed host A is finally excluded from the reading set. Simultaneously, we can observe that the maximum local synchronization error is never lower than the real synchronization error.

8 Summary

In future, we will have to build critical systems for which it will be very difficult to assume that the underlying system behaves like a synchronous system. For example, we will need to support wireless communication links, commercial off-the-shelf components without guaranteed response times and applications that dynamically change their processor and network load.

We have introduced the specification of adaptive clock synchronization which permits implementations to cope with potentially unbounded number of message delays and process failures. Specifically, we have proposed a clock synchronization algorithm which allows every process to calculate a local view of the maximum synchronization error $E_p(T)$ which defines an upper bound on the deviation of the local clock from the correct clocks in the system.

The local maximum synchronization error $E_p(T)$ can be used by the higher level applications or systems to adapt their behavior to the potential faults occurring at (or below) the clock synchronization level. Specifically, we have shown how the maximum clock synchronization error can be used to locally detect the growing imprecision.

We believe that the new specification of the clock synchronization algorithm allows for construction of safer systems. Systems using the adaptive clock synchronization algorithm as an underlying layer, will no longer be exposed to the clock synchronization layer crashes whenever the strong clock synchronization algorithm assumptions are violated.

References

[1] E. Anceaume and I. Puaut. Performance evaluation of clock synchronization algorithms. Technical Report 3526, INRIA, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France), October 1998.

[2] K. Arvind. Probabilistic clock synchronization in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 05(5):474–487, 1994.

[3] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, September 1989.

[4] F. Cristian and C. Fetzer. Fault-tolerant internal clock synchronization. In *Proceedings of the Thirteenth Symposium on Reliable Distributed Systems (SRDS1994)*, pages 22–31, Dana Point, Ca., Oct 1994.

[5] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, pages 642–657, June 1999.

[6] R. Fan and N. A. Lynch. Gradient clock synchronization. *Distributed Computing*, 18(4):255–266, 2006.

[7] C. Fetzer and F. Cristian. An optimal internal clock synchronization algorithm. In *Proceedings of the 10th Annual IEEE Conference on Computer Assurance (COMPASS1995)*, pages 187–196, Gaithersburg, MD, June 1995.

[8] C. Fetzer and F. Cristian. A fail-aware membership service. In *Proceedings of the 16th Symposium on Reliable Distributed Systems (SRDS1997)*, pages 157–164, Oct 1997.

[9] C. Fetzer and F. Cristian. Integrating external and internal clock synchronization. *Journal of Real-Time Systems*, 12(2):123–171, March 1997.

[10] C. Fetzer and F. Cristian. Building fault-tolerant hardware clocks. In *Proceedings of the Seventh IFIP International Working Conference on Dependable Computing for Critical Applications*, pages 59–78, San Jose, USA, Jan 1999.

[11] C. Fetzer and F. Cristian. A fail-aware datagram service. *IEE Proceedings - Software Engineering*, pages 58–74, April 1999.

[12] C. Fetzer and F. Cristian. Fail-awareness: An approach to construct fail-safe applications. *Journal of Real-Time Systems*, pages 203–238, March 2003.

[13] FlexRay Consortium. *FlexRay Communications System Protocol Specification*, version 2.1 revision a edition, December 2005.

[14] Z. Jerzak, R. Fach, and C. Fetzer. Fail-aware publish/subscribe. In *Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007)*, pages 113–125, Cambridge, MA, USA, July 2007. IEEE Computer Society.

[15] H. Kopetz and G. Grunsteidl. TTP – A Protocol for Fault-Tolerant Real-Time Systems. *Computer*, 27(1):14–23, 1994.

[16] H. Kopetz, A. Kruger, D. Millinger, and A. Schedl. A synchronization strategy for a time-triggered multicluster real-time system. In *14th Symposium on Reliable Distributed Systems, 1995. Proceedings*, pages 154–161, Bad Neuenahr, Germany, September 1995.

[17] B. Liskov. Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, 6(4):211–219, 1993.

[18] H. Moser and U. Schmid. Optimal clock synchronization revisited: Upper and lower bounds in real-time systems. *Principles of Distributed Systems*, 4305:94–109, November 2006.

[19] F. Schmuck and F. Cristian. Continuous clock amortization need not affect the precision of a clock synchronization algorithm. In *PODC '90: Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 133–143, New York, NY, USA, 1990. ACM.

[20] A. Varga. Using the omnet++ discrete event simulation system in education. *IEEE Transactions on Education*, 42(4):372, 1999.

[21] P. Veríssimo, L. Rodrigues, and A. Casimiro. CesiumSpray: a precise and accurate global time servicefor large-scale systems. *Real-Time Syst.*, 12(3):243–294, 1997.

[22] U. Wappler and C. Fetzer. Hardware failure virtualization via software encoded processing. In *5th IEEE International Conference on Industrial Informatics (INDIN 2007)*, volume 2, pages 977–982, June 2007.

[23] J. L. Welch and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computing*, 77(1):1–36, 1988.

[24] J. Widder and U. Schmid. Booting clock synchronization in partially synchronous systems with hybrid process and link failures. *Distributed Computing*, 20(2):115–140, May 2007.