Fault-Tolerant External Clock Synchronization

Flaviu Cristian and Christof Fetzer
Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093–0114*
E-Mail: {flaviu, cfetzer}@cs.ucsd.edu

Abstract

We address the problem of how to integrate fault-tolerant internal and external clock synchronization. We propose a new algorithm which provides both external and internal clock synchronization for as long as no more than F reference time servers out of a total of $2F+1$ are faulty. When the number of faulty reference time servers exceeds F, the algorithm degrades to a fault-tolerant internal clock synchronization algorithm. We prove that at least $2F+1$ reference time servers are necessary for achieving external clock synchronization when up to F reference time servers can suffer arbitrary failures, thus our algorithm provides maximum fault-tolerance. The algorithm is also optimal in another sense: we show that the maximum deviation between reference time and the clocks of non-reference time servers is minimal.

1 Introduction

Synchronized clocks are crucial for many fault-tolerant, real-time systems. Clocks can be externally or internally synchronized [1]: *externally synchronized* clocks are within a constant interval around reference time while *internally synchronized* clocks are within a constant interval of each other. This implies that externally synchronized clocks are always internally synchronized, but internally synchronized are not always externally synchronized.

To achieve external synchronization, at least some time servers must have physical access to reference time. Such servers typically use radio receivers to receive the time signal broadcasts by a standard source of time such as GPS. A time server with access to reference time signals is a *reference time server*, the other servers are *non-reference time servers*. Since the cost of the hardware components needed to access external reference time signals is not negligible, one typically tries to minimize the number of reference time servers needed in a given system. Our first goal is to illuminate the issue of how many reference servers are necessary for providing external synchronization when up to F reference servers can fail in arbitrary ways. We prove that a total of $2F + 1$ reference time servers is necessary and sufficient.

In most fault-tolerant real-time systems we know, internal synchronization is required for the correct operation of the system, while external synchronization is mainly used to give users convenient reference time information. For example in the Advanced Automation System, the correctness of automated recovery from component failures depends on correct internal clock synchronization [2] but the system can continue to function in the absence of external clock synchronization in a degraded way. Our second goal is to present an algorithm that maintains externally synchronized clocks for as long as at least a majority of reference time servers remains correct, and degrades to internal clock synchronization when half or more of the reference servers become faulty. In particular, in a practical system with only one reference time server, our algorithm will guarantee that the correct clocks will always be internally synchronized and that they will also stay externally synchronized for as long as the reference time server stays correct.

External clock synchronization algorithms typically bound the internal deviation between clocks by at least twice the reading error that a reference time server makes in reading the reference time [1]. Our third goal is to show how one can bound the maximum deviation between internally synchronized clocks independently of this reading error. For example, in our

*This work was partially supported by grants from the Air Force Office of Scientific Research, the German Academic Exchange Service (DAAD), the Powell Foundation, Sun Microsystems, and the Microelectronics Innovation and Computer Research Opportunities in California

Dependable Systems Laboratory, access to reference time involves a maximum error of more than 10 ms because the access is via a serial interface. However, instead of resigning to have a maximum deviation between clocks of 20ms, we want to bound their distance by a constant *smaller* than 10ms. Our algorithm solves the above problem by providing a maximum deviation between clocks that is independent of the maximum error in accessing reference time. Thus, we achieve tight internal synchronization even when the reference time servers are only loosely synchronized to reference time. Other remarkable properties of our algorithm are as follows: the maximum deviation from external reference time is provably minimum and the drift rate from real-time when in degraded mode is also provably minimum (about twice the maximum drift rate of hardware clocks).

2 System Model

We consider distributed systems consisting of nodes connected by a communication network. The processes that run on network nodes communicate with each other by exchanging messages. Each node runs a *time server* whose goal is to provide a synchronized clock to local clients. We partition the set of time server processes $\mathcal{P}$ into the set of reference time servers $\mathcal{P}_R$ and the set of non-reference time servers $\mathcal{P}_N$. Let $N \triangleq |\mathcal{P}|$, $N_R \triangleq |\mathcal{P}_R|$, and $N_N \triangleq |\mathcal{P}_N|$ denote the number of time servers, reference time servers, and non-reference time servers, respectively. We assume that each reference time server and each non-reference time server has a unique rank, denoted by function *rank*, such that for $r \in \mathcal{P}_R$, $rank(r) \in \{1, .., N_R - 1\}$ and for $p \in \mathcal{P}_N$, $rank(p) \in \{1, .., N_N - 1\}$. We sometimes use the term *process* to refer to a time server.

2.1 Clocks

We assume that the granularity of real-time is infinitesimal and denote with $\mathcal{RT}$ the set of all real-time values. We also assume that with each event e, one can associate its point of occurrence in real-time $to(e)$ [7]. Reference time is a granular representation of real-time. It is defined by a sequence of ticks tr_i, $i \in \{0, 1, 2, \ldots\}$, such that (1) the distance between two successive ticks is exactly g_r (the granularity of reference time) and (2) real-time and reference time are in perfect synchrony:

$$\forall i : g_r = to(tr_{i+1}) - to(tr_i) \wedge tr_i = to(tr_i).$$

For simplicity, we assume that the granularity of reference time is negligible and we will not distinguish between reference time and real-time.

Each time server has access to a *local hardware clock*. We assume that the granularity G of such clocks is negligible and denote with $\mathcal{CT}$ the set of clock values. A hardware clock typically consists of an oscillator and a counting register that is incremented by the ticks of the oscillator. For any time server $p \in \mathcal{P}$, $H_p(t)$ denotes the value displayed by the counter of p's hardware clock at real-time t. A hardware clock H_p is *correct* in a time interval $[s, v]$ if it measures the passage of all real-time intervals $[t, u] \subseteq [s, v]$ with an error of at most $\rho(u-t)+G$, where ρ is the maximum hardware clock drift rate specified by the clock manufacturer:

$$(1\text{-}\rho)(\text{u-t})\text{-G} \leq H_p(\text{u})\text{-}H_p(t) \leq (1+\rho)(\text{u-t})+\text{G}.$$

For most quartz clocks available in modern computers, the constant ρ is of the order of 10^{-6}. Since ρ is such a small quantity, in this paper we ignore terms ρ^i for $i \geq 2$. For example, we equate $(1+\rho)^{-1}$ with $(1-\rho)$ and $(1-\rho)^{-1}$ with $(1+\rho)$.

Reference clocks allow reference time servers to access reference time. By definition, a correct reference clock is within a constant interval of reference time. This implies that the drift rate of a correct reference clock is zero. We represent the reference clock of a reference time server p by a mapping $X_p : \mathcal{RT} \rightarrow \mathcal{CT}$ and denote with $X_p(t)$ the value of p's reference clock at real-time t. A reference clock X_p is *correct* at time t if the deviation between real-time and X_p is bounded by an a priori given constant Δ:

$$|X_p(t) - t| \leq \Delta.$$

Reference clocks can be implemented in various ways. For example, in our Dependable Systems Laboratory two workstations have access to a GPS-receiver via a serial interface and a thread in each reference time server p periodically reads the current time provided by the GPS-receiver to determine the difference d between reference time and p's hardware clock. The current value of p's reference clock is defined by the sum of the current value of p's hardware clock plus the last computed difference value d. In our system the maximum deviation Δ between a reference clock and reference time is more than 10 milliseconds. A hardware implementation of reference clocks could reduce this difference to a few microseconds.

Clock synchronization algorithms do not directly synchronize the local hardware clocks of network nodes. Instead, they introduce the concept of *virtual clocks* maintained by time servers. Our goal will be to ensure that virtual clocks are both internally and externally synchronized. In particular, each reference

time server provides a virtual clock for its local clients instead of maintaining only a reference clock. This is necessary, to provide internally synchronized clocks with (1) a maximum deviation independent of Δ, and (2) irrespective of the number of failed reference time servers. The virtual clock of time server $p \in \mathcal{P}$ is represented by a mapping $C_p : \mathcal{RT} \rightarrow \mathcal{CT}$. The value $C_p(t)$ is determined by adding to the local hardware clock $H_p(t)$ a periodically updated *adjustment value.*

2.2 Remote Clock Readings

Each time server has to estimate the values displayed by all other virtual clocks and reference clocks, to adjust its own virtual clock. This estimation is performed by making use of a *remote clock reading method.* The reading method allows each time server p to estimate the value displayed by a remote virtual clock C_q when p's own virtual clock reads some given time T. We use $\mathcal{C}_q(T,p)$ to denote p's approximation of q's clock and adopt the convention of denoting t the real-time when p's virtual clock reads T: $T = C_p(t)$. We assume that the remote clock reading error provided by the reading method above is bounded by an a priori given constant Λ, to be called the *maximum clock reading error*: *for any two time servers p and q correct at t, the clock reading error is bounded by*

$$|C_q(t) - \mathcal{C}_q(T,p)| \leq \Lambda.$$

The reading method can also be used to estimate remote reference clocks. We denote with $\mathcal{X}_q(T,p)$ the approximation of reference clock X_q by a time server p when p's local time is T. We assume that the reading error for remote reference clocks is bounded by the same constant Λ: *for any time server p and any reference time server q both correct at t, the clock reading error is bounded by*

$$|X_q(t) - \mathcal{X}_q(T,p)| \leq \Lambda.$$

The execution of the reading method takes time. We assume that any remote reading terminates in at most RD clock time units. In other terms, $RD(1+\rho)$ real-time units are sufficient to read any remote virtual or reference clock. We assume that each process can read its hardware clock and each reference timer server can read its reference clock with a *negligible* reading error.

2.3 Failure Model

We assume that the failures experienced by hardware clocks, reference clocks and processes are *arbitrary.* By definition, a correct process has a correct hardware clock and a correct reference time server has a correct reference clock. For simplicity, we assume that failed processes do not recover. Thus, we do not discuss the reintegration of recovered processes. We also assume that communication is reliable, so that the reading method (which depends on messages for reading remote clocks) always provides a reading error smaller than Λ. This avoids discussing clock reading failures which occur when a correct time server p estimates a remote virtual or reference clock of a correct time server q with an error greater than Λ. Methods for making synchronization algorithms tolerant to remote clock reading failures are discussed in [4].

3 Requirements

After introducing our system model, we can define internal and external clock synchronization more precisely. Let t^0 denote the earliest point in real-time for which the virtual clocks must be internally synchronized. The first objective of an internal clock synchronization algorithm is to bound the deviation between any two correct virtual clocks by a constant δ to be called the *maximum internal deviation*: *for any two processes p and q that are correct at real-time $t \geq t^0$:*

$$|C_p(t) - C_q(t)| \leq \delta.$$

The second goal is to bound the drift rate of virtual clocks from real-time by a constant $\rho_v \ll 1$. The maximum discontinuity, D, of virtual clocks accounts for the granularity and the adjustments of virtual clocks. Thus, the second requirement is expressed as follows: *for any process p that is correct in interval $[t,u]$, where $t^0 \leq t \leq u$:*

$$(1\text{-}\rho_v)(\text{u-t})\text{-D} \leq C_p(\text{u})\text{-}C_p(\text{t}) \leq (1+\rho_v)(\text{u-t})+D.$$

An internal clock synchronization algorithm is *correct* when it satisfies both of the above requirements.

A correct external clock synchronization algorithms has to bound the deviation between a correct virtual clock and reference time by an a priori given constant ϵ, to be called the *maximum external deviation*: *for any process p that is correct at time t,*

$$|C_p(t) - t| \leq \epsilon.$$

4 Lower and Upper Bounds

In this section we derive a lower and an upper bound for the number of reference time servers needed to provide externally synchronized clocks. We assume that there exists at least one correct non-reference time server, and that, at any time, *the number of faulty reference time servers is bounded by F (FA1).*

In systems without access to reference time and without message authentication at least $3F+1$ time

servers are needed to guarantee internal clock synchronization despite F arbitrary time server failures [5]. We show below that at least $2F+1$ reference time servers are needed to guarantee that the virtual clocks of non-reference time servers are externally synchronized. The reason for the difference in the number of time servers needed to mask F failures in internal and external synchronization is the following: (1) Internal clock synchronization without access to reference time servers requires a protocol similar to a Byzantine agreement protocol which requires $3F+1$ nodes. (2) Correct reference time servers approximately agree on the current time by receiving external reference time signals. This allows non-reference time servers to agree approximately on the current time without the need to communicate between each other.

Note that the $2F+1$ lower bound is not necessarily valid for systems in which the failure semantics of reference time servers is stronger. For example, if the reference time servers can only crash, the tight lower bound for the total number of reference time servers needed to provide external clock synchronization despite F faulty reference time servers would be $F+1$. To understand why this is so, let us assume that the clock reading method returns an error when a process tries to read the reference clock of a crashed reference time server (in our system model this can be easily implemented). Then $F+1$ reference servers are sufficient to guarantee external synchronization of non-reference time servers. Indeed, at any time t, at least one reference time server is correct and each non-reference time server p can periodically read all reference clocks and use the reading of a non-crashed reference time server to adjust its virtual clock. $F+1$ is a lower bound because, if the total number of reference servers is F and all F can crash, a non-reference time server obviously cannot synchronize its clock to reference time.

4.1 Upper Bound

We first show that $2F+1$ is an upper bound for the number of reference time servers needed to guarantee external synchronization despite F faulty reference servers by proposing an algorithm ECS which achieves external synchronization for $2F+1$ reference time servers. The intuition behind ECS is as follows. In this algorithm, each non-reference server p reads all reference clocks and then sorts these approximations in increasing order in an array X[0::N_R-1]. Amongst the $F+1$ smallest reference clock values in X there must be correct reference clock value, say in position $l \in [0, F]$. Similarly, amongst the highest $F+1$ reference values in X there is a correct one, say in position m. Since both the values X[l] and X[m] are correct, in the sense of being within a given constant of reference time, it follows that any value of the array *between* positions l and m, in particular $X[F]$, is also correct. Thus, if each process p sets its virtual clock to the $F+1$th smallest reference clock reading, all virtual clocks will be externally synchronized.

```
(1)   Time  A = 0, X[N_R], T;
(2)   function Time C() { return H() + A; }
(3)   function void synchronizer () {
(4)         forever {
(5)             T = C() + RD;
(6)             parallel { ∀q ∈ P_R :
                    X[rank(q)] = X_q(T) }
(7)             sort(X);
(8)             A = A + X[F] − T  } }
```

Figure 1: External clock synchronization algorithm ECS

The detailed algorithm is given in Figure 1 in a C-like notation. The virtual clock C_p of p is implemented by the function C (line 2). It is defined by the sum of the current value of p's hardware clock (returned by the function $H()$) and the current value of the adjustment variable A. Process p adjusts its virtual clock by recalculating its adjustment variable A in an endless loop (lines 4-8). It estimates the clocks of all reference time servers for the time given by variable T. By hypothesis, the constant RD denotes the maximum clock-time it takes to execute the remote clock reading of all reference clocks performed in line 6. The estimations of the reference clocks are computed in parallel and stored in array X (line 6). Function *rank* assigns to each reference time server a unique number between 0 and $N_R - 1$, where N_R is the total number of reference time servers. The function call $sort(X)$ sorts the entries of array X (line 7) in non-decreasing order. In line 8 the adjustment variable is updated such that the value of p's virtual clock is changed from T to $X[F]$. If the adjustment value A is updated at least every r_{max} real time units, the maximum external deviation of virtual clocks provided by the ECS algorithm is at most $\epsilon \triangleq \Lambda + \Delta + \rho r_{max}$.

Theorem (T1): *For any process p correct at t that executes algorithm ECS:*

$$|C_p(t) - t| \leq \epsilon$$

The proof of this theorem and all other theorems can be found in [3].

4.2 Lower Bound

Lower Bound Theorem (LB): *There is no correct external clock synchronization algorithm capable of masking F faulty reference time servers if the total number of reference time servers is not greater than $2F$.*

The intuition behind the the *lower bound theorem* is as follows. When less than $2F+1$ reference time servers are available, for any synchronization algorithm that claims to achieve external synchronization it is possible to find two indistinguishable executions in which the set of reference time servers is partitioned into two sets P and Q with each containing at most F reference time servers. In one execution P contains all correct reference time servers and Q contains all faulty ones, while in the other execution the situation is reversed. The executions can be chosen such that the reference clocks in each set always show the same value, but there is a small drift between two reference clocks that belong to the different sets P and Q. For example, let the drift rate dr of all hardware clocks in the system be the same, let the drift rate of the reference clocks in P be dr, and let the drift rate of the reference clocks in Q be $dr+\rho$. Set dr=0 in the first execution, i.e. the reference time servers in P are correct and the ones in Q are faulty. In the second execution let dr=-ρ, i.e. the reference time servers in P are faulty and the ones in Q are correct. The deviation between the reference clocks of P and Q will eventually be greater than any given bound, thus each non-reference time server p has eventually to decide to which of the two sets it should synchronize its clock. We show that this decision cannot always be done correctly: if the decision is is to synchronize towards P and the algorithm actually experiences the second execution, i.e. the decision is wrong, similarly, if the decision is towards Q and the algorithm experiences the first execution the decision will be equally wrong. Thus, less than $2F+1$ reference clocks are not enough for achieving external clock synchronization despite up to F faulty reference clocks.

5 Optimal External Clock Synchronization

We now give lower bounds for the maximum external deviation ϵ, the maximum discontinuity G and the drift rate ρ_v of externally synchronized clocks. Because of space restrictions we have to limit ourselves to give the lower bounds without the intuitions and proofs. More details can be found in [3].

First, we give a lower bound for the maximum external deviation achievable by any external clock synchronization algorithm A. We use the following assumptions in the derivation of this bound:

1. A virtual clock of a correct non-reference time server p is adjusted at least every r_{max} time units, that is, in each execution of A, p's virtual clock is updated at least every r_{max} time units and there exists an execution for which it is adjusted exactly every r_{max} time units.

2. Constants Δ and Λ are tight: we can assume that there exists an execution for which the reference clocks are Δ apart from real-time and all clock reading errors are Λ.

Theorem (LB1): *The maximum external deviation of virtual clocks of non-reference time servers is at least* $\epsilon_{opt} \triangleq \Delta + \Lambda + \rho r_{max}$.

Definition: An external clock synchronization algorithm A is called *optimal*, if the maximum external deviation achieved by A is $\epsilon_{opt} = \Delta + \Lambda + \rho r_{max}$.

Second, we show that the drift rate of externally synchronized clocks is zero, i.e. $\rho_v = 0$. Let non-reference time server p be correct at times t, s such that $t > s$. By requirement *external deviation*: $|C_p(t) - t| \leq \epsilon$ and $|C_p(s) - s| \leq \epsilon$, thus:

$$\begin{aligned}(t-s) - 2\epsilon &\leq (t-\epsilon) - (s+\epsilon) \leq C_p(t) - C_p(s) \\ &\leq (t+\epsilon) - (s-\epsilon) = (t-s) + 2\epsilon.\end{aligned}$$

This shows that the drift rate of externally synchronized clocks is zero. We also derive a lower bound for the maximum discontinuity:

Theorem (LB2): *The maximum discontinuity of externally synchronized clocks is at least* $G_o \triangleq 2\epsilon_{opt}$.

All externally synchronized clocks are also internally synchronized, since the deviation between the virtual clocks of two correct non-reference time servers, the maximum discontinuity and the drift rate are bounded by constants. The deviation between two externally synchronized clocks is trivially bounded by 2ϵ, i.e. $\delta \leq 2\epsilon$. Furthermore, we showed that the drift rate ρ_v of externally synchronized clocks is zero and the maximum discontinuity G is at most 2ϵ.

6 Gracefully Degradable External Clock Synchronization

We propose a clock synchronization algorithm CS that achieves external clock synchronization for as

long as less than half of the reference time servers suffer arbitrary failures and degrades to providing only internal clock synchronization when half or more of the reference time servers suffer arbitrary failures.

The intuition behind the proposed algorithm is as follows. The virtual clocks are synchronized in rounds at least every r_{max} time units and initially they are at most $\Delta + \Lambda$ apart from real-time. Since the lenght of a round is bounded by r_{max} real-time units, a hardware clock can drift from real-time by at most ρr_{max} during one round. The midpoint of the interval containing exactly all correct virtual clocks at the end of a round is therefore at most $\Delta + \Lambda + \rho r_{max}$ apart from real-time. Processes use an approximation AP of this midpoint which is also at most $\Delta + \Lambda + \rho r_{max}$ apart from real-time. Furthermore, processes can approximate real-time with an error of at most $\Delta + \Lambda$. Therefore, a non-reference time server p only needs to adjust its AP towards its approximation of real-time by at most ρr_{max} to guarantee that p's clock when set to the modified AP is at most $\Delta + \Lambda$ apart from real-time. This allows optimal external synchronization when a majority of the reference time servers is correct. The maximum internal deviation of the clocks is always close to the best possible deviation achievable by an internal clock synchronization algorithm, because approximation AP is only slightly changed and therefore the errors introduced by too many defect reference time servers is limited.

6.1 Algorithm CS

Figure 2 shows algorithm CS which is executed by each non-reference time server. The virtual clock of any time server is implemented by the function C. This defines a virtual clock as the sum of the hardware clock and the value of an adjustment variable A. The algorithm is initialized by the function *init* (line 5-8). The function *InitialAdjustement* determines the initial adjustment value A and the local time T at which the virtual clock has to be adjusted for the first time. All processes agree on the local times when they synchronize their clocks: correct processes synchronize their clocks at local times $T, T+P, T+2P, \ldots$. The adjustments of the virtual clocks is done by the function *synchronizer*. To bound the maximum external and internal deviation, the execution of the *synchronizer* function is scheduled every P (local) time units. We denote the maximum real-time distance between two successive adjustments by constant r_{max}. The first execution of the function *synchronizer* is scheduled at time $T - RD$ (line 7), since the reading of remote clocks can take up to RD time units.

```
(1)   Time  A = 0, T;
(2)   function Time C() { return H() + A; }
(3)   function void init () {
(4)       (A,T) = InitialAdjustement();
(5)       schedule (synchronizer, P, T − RD); }
(6)   function void synchronizer() {
(7)       Time X[N_R], Y[N_N], midX, midY;
(8)       Time M = ρ * r_max;
(9)       parbegin {
(10)          ∀r ∈ P_R : X[rank(r)] = C_r(T);
(11)          ∀p ∈ P_N : Y[rank(p)] = C_p(T);
(12)      } parend
(13)      sort (X); sort (Y);
(14)      midX = (X[F_R]+X[N_R−F_R−1])/2;
(15)      midY = 0.5min{T-Λ,Y[F_N]} +
                 0.5max{T+Λ,Y[N_N − F_N − 1]};
(16)      if (|midX − midY| < M)
(17)          A = A + midX - T;
(18)      else
(19)          A=A+midY+sig(midX-midY)*M-T;
(20)      T = T + P;  }
```

Figure 2: Clock Synchronization Algorithm CS

The function *synchronizer* reads all virtual and reference clocks in parallel (line 9-12). The remote clock reading function C_p estimates p's virtual clock of the current round. The estimations of the virtual clocks and the reference clocks at time T are stored in arrays Y and X. Arrays Y and X are sorted in non-decreasing order by a call to function *sort* (line 13). Variable $midX$ is set to the fault-tolerant midpoint [8] of array X (line 14). Therefore, $midX$ is at most $\Lambda+\Delta$ apart from real-time at time T whenever at most F of the reference time servers are faulty. Variable $midY$ is set to the Λ extended fault-tolerant midpoint function introduced in [4] to decrease the distance between $midY$ and real-time.

We will show that the value of $midY$ is at most $\Lambda + \Delta + \rho r_{max}$ apart from reference time at time T. The adjustment towards $midX$ can therefore be restricted by ρr_{max} to guarantee a maximum external deviation of $\Lambda + \Delta + \rho r_{max}$ (line 16-20). The signum function *sig* returns 1 when its argument is positive, zero when its argument is zero, and -1 when its argument is negative (line 19). Variable T is adjusted for the next execution of *synchronizer* in line 20.

6.2 Analysis of Algorithm CS

To simplify the analysis of algorithm CS, let us assume that the estimation of remote clocks (line 9-12) terminates when the local virtual clock shows T and that the execution time of the remaining statements of the function *synchronizer* is negligible. We represent the values taken in time by the variables of algorithm CS by sequences. The points in real-time when process p adjusts its clock is denoted by sequence (t_p^k). Time t^k denotes the real-time when all correct processes have just synchronized their virtual clocks: $t^k \triangleq max\{t_p^k \mid p \text{ correct at time } t_p^k\}$. For round k and a correct process p, the adjustment value computed for variable A is denoted by A_p^k. Process p's k+1-th round starts when process p's adjustment variable is changed from A_p^k to A_p^{k+1} at time t_p^{k+1}. At the end of every round all correct processes try to estimate the values of all clocks. The rounds of different processes can overlap: a process can start round $k+1$ while another process is still in round k. Process p always approximates all remote virtual clocks with respect to the adjustment values of its current round k. To capture this, we use the well-known concept of a round clock: process p*'s round clock* for k is defined as

$$C_p^k(t) \triangleq H_p(t) + A_p^k. \quad (1)$$

Process p's virtual clock $C_p(t)$ in interval $[t_p^k, t_p^{k+1}[$ is defined by p's clock for round k:

$$C_p(t) \triangleq C_p^k(t) \text{ for } t_p^k \leq t < t_p^{k+1}. \quad (2)$$

Process p's sorted estimations of the reference clocks and the k-th round clocks at the end of round k are represented by the arrays of clock values X_p^k and Y_p^k. The values of variables T, $midX$, and $midY$ at the end of round k of process p are denoted by T_p^k, $midX_p^k$, and $midY_p^k$, respectively.

To prove the correctness of algorithm CS we assume that the clocks are initially synchronized and we constrain the interleaving of synchronization rounds. Furthermore, we have to constrain the number of faulty time servers.

6.2.1 Initialization

We require that, at the start of round $k = 0$, the external deviation be bounded by $\Lambda + \Delta$ and the internal deviation be bounded by δ_S, where δ_S denotes the maximal internal deviation between correct clocks at the start of a new round (see section 6.4). *For any processes p,q correct at real-time* t^0,

$$|C_p(t^0) - t^0| \leq \Lambda + \Delta \ \wedge \ |C_p(t^0) - C_q(t^0)| \leq \delta_S \ .$$

6.2.2 Interval Constraints

To bound the maximum internal and external deviation, the length of a clock synchronization round must be bounded: *For any process p that is correct at* t_p^{k+1},

$$r_{min} \leq t_p^{k+1} - t_p^k \leq r_{max}.$$

The real-time delay between the beginning of the same synchronization round for different processes must also be bounded; *For any processes p and q correct at* t_p^k *and* t_q^k, *respectively,*

$$|t_p^k - t_q^k| \leq \beta.$$

The overlap of rounds is restricted by requiring that

$$\beta \leq r_{min}.$$

6.2.3 Failure Assumptions

We analyze algorithm CS for two failure hypotheses $FS1$ and $FS2$:

(FS1): The number of non-reference time servers suffering arbitrary failures is bounded by F_N and the number of reference servers suffering arbitrary failures is bounded by F_R.

(FS2): The number of non-reference servers suffering arbitrary failures is bounded by F_N.

We assume that the number of reference time servers N_R is at least $2F_R + 1$ for hypothesis (FS1), while N_R is not constrained for failure assumption (FS2). The number of non-reference time servers is required to be at least $3F_N + 1$ [5].

6.3 Analysis for Failure Hypothesis FS1

Lemma (L1): $\forall k \forall p \in \mathcal{P}_N$: p *correct at* $t_p^{k+1} \rightarrow |midX_p^k - t_p^{k+1}| \leq \Delta + \Lambda$.

Lemma (L2): $\forall k \forall p \in \mathcal{P}_N$: p *correct at* $t_p^k \rightarrow |C_p^k(t_p^k) - t_p^k| \leq \Delta + \Lambda$.

We can conclude from lemma (L2) that algorithm CS is correct and that the maximum external deviation is bounded by $\epsilon \triangleq \Lambda + \Delta + \rho r_{max}$.

Theorem (T2): *For any non-reference time server p correct at time t:*

$$|C_p(t) - t| \leq \epsilon.$$

6.4 Analysis for Failure Assumptions FS2

We now show that the internal deviation of virtual clocks remains bounded even if half or more of the reference time servers suffer arbitrary failures. The lower bound theorem shows that external synchronization cannot be guaranteed for failure assumptions FS2. We

show that the maximum internal deviation is always bounded by $\delta \triangleq 4\Lambda + 9\rho r_{max} + 2\rho\beta$ by first showing that the maximum deviation at the start of a round is bounded by $\delta_S \triangleq 4\Lambda + 6\rho r_{max} + 2\rho\beta$.

Lemma (L3): *For any non-reference time servers p and q that are correct at t^k:*

$$|C_p(t^k) - C_q(t^k)| \leq \delta_S.$$

From Lemma (L3) we can prove that the maximum internal deviation is bounded by δ.

Theorem (T3): *For any non-reference time servers p and q correct at t:*

$$|C_p(t) - C_q(t)| \leq \delta.$$

7 Conclusion

We explored the problem of integrating internal and external clock synchronization by a single algorithm: how to provide external clock synchronization when a *sufficient number* of reference time servers are correct and degrade to internal clock synchronization when the number of faulty reference clocks becomes too large. We derived a lower bound for the number of reference time servers needed to achieve external synchronization: at least $2F+1$ reference time servers are needed to mask F arbitrarily faulty reference time servers.

We derived lower bounds for the external deviation ϵ, the drift rate of virtual clocks ρ_v and the maximum discontinuity G achievable by any external clock synchronization algorithm. The proposed fault-tolerant external clock synchronization algorithm is optimal in the sense that it achieves the provably minimum external deviation, drift rate, and maximum discontinuity whenever more than half of the reference time servers are correct. This algorithm is also optimal with respect to the number of reference time servers needed to guarantee external clock synchronization and the drift rate of virtual clocks when at least half of the reference time server suffer arbitrary failures. The bound for the internal deviation $(4\Lambda + 9\rho r_{max} + 2\rho\beta)$ is very close to the optimal maximum internal clock synchronization of $4\Lambda+4\rho r_{max}$ [6] achievable by a convergence function based internal clock synchronization algorithm [9]. The drift rate of the virtual clocks is in this case bounded by two times the maximum drift rate of hardware clocks, i.e. $\rho_v \leq 2\rho$.

When a majority of the reference time servers is correct, the proposed external clock synchronization algorithm achieves the following properties. The maximum external deviation is bounded by $\epsilon = \Lambda + \Delta + \rho r_{max}$, the drift rate ρ_v is zero, and the maximum discontinuity G is 2ϵ. The maximum internal deviation δ is independently bounded by $\delta \leq 2\epsilon$ and $\delta \leq 4\Lambda + 9\rho r_{max} + 2\rho\beta$. Thus, if the error in reading reference clocks is large, a tight internal synchronization is still achievable.

References

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

[2] F. Cristian, B. Dancey, and J. Dehn. Fault-tolerance in the Advanced Automation System. In *Proceedings of the Twentieth Symposium on Fault-Tolerant Computing*, pages 6–17, Newcastle-upon-Tyne, UK, Jun 1990.

[3] F. Cristian and C. Fetzer. Fault-tolerant external clock synchronization. Technical Report CS94-378, Dept of Computer Science and Engineering, University of California, San Diego, La Jolla, CA, 1994.

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

[5] D. Dolev, J. Y. Halpern, and R. Strong. On the possibility and impossibility of achieving clock synchronization. *Journal of Computer and System Science*, 32(2):230–250, 1986.

[6] C. Fetzer and F. Cristian. An optimal internal clock synchronization algorithm. In *Proceedings of the 10th Annual IEEE Conference on Computer Assurance*, Gaithersburg, MD., June 1995.

[7] H. Kopetz and W. Ochsenreiter. Interval measurements in distributed real time systems. In *7th International Conference on Distributed Computing Systems*, pages 292–298, Sept. 1987.

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

[9] F. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report 87-859, Dept of Computer Science, Cornell University, Aug 1987.