On the Accuracy and Efficiency of External Clock Estimation

Daniela Tulone *

Dipartimento di Informatica, Universita' di Pisa
via F.Buonarroti 2, Pisa 56127, Italy
tulone@di.unipi.it

Abstract. We propose a general primitive that improves the accuracy of a *local clock reading* without additional inputs, and show how this leads to an improvement of the Fetzer and Cristian *optimality result* for external clock synchronization using the same assumptions.
We apply our primitive to a problem *more general* than the clock synchronization problem and suitable for very large and dynamic populations. We address the problem of providing processes dispersed in a WAN and requiring time information, with a *close approximation* of the reference time, and propose an efficient protocol built on top of our reading clock method. Our approach is based on Quorum Systems and is resilient to malicious failures. The impact that our *time service* has on applications is large-scale (i.e e–commerce applications, access control, time stamping).

1 Introduction

Accurate, reliable time is crucial for many fault–tolerant real–time systems involving financial and legal transactions, transportation and many other applications employing widely distributed resources. Since hardware clocks drift from their *reference time* and this deviation can grow unbounded, distributed systems requiring time information need to be endowed with a mechanism that guarantees an *a priori known* bound for their *external deviation*. This problem is well–known as *external clock synchronization* problem and is solved through periodic adjustments of the clock to an *external reference source*. During the last 15 years, a number of solutions have been devised to guarantee external clock synchronization of a set of distributed nodes, and the refined *time accuracy* has reflected an increased demand due to the improved speed in the networks and operating systems. All these solutions share a common viewpoint: they improve the time accuracy by refining the periodic *clock adjustments*. This is achieved in different ways: protocols such as [11][10][15][3] exploit information regarding the hardware drift rate, Cristian [2] and recently Casimiro et al. [7] analyze the *remote clock reading error*, and other protocols such as [9][15][1] focus on the *convergence function*. To our knowledge nothing has been done to improve the time accuracy of a clock between two consecutive adjustments: reading a *local clock* has been referred in the literature as a simple *function reading* whose associate error bound grows proportionally to its maximum drift rate and to the time elapsed since its last synchronization. This *error growth* appears also in Fetzer and Cristian optimality result for external clock synchronization [1]: they set the lower bound for the maximum deviation of a clock synchronized at least every Γ time units, to $\Lambda + \Delta + \rho\Gamma$ with Δ error relative to the external reference source, Λ remote clock reading error and $\rho\Gamma$ maximum error growth between adjustments. Our target is to improve the precision of a clock by reducing the maximum error growth $\rho\Gamma$. We achieve this by exploiting the assumption, present in all discussions on clocks, that hardware clocks are within a linear envelope of the real time. Therefore, we *do not* make more restrictive assumptions on the behavior of the hardware drift rate or exploit a more realistic pattern as our discussion is valid for any distribution of the hardware drift rate. The study of the hardware drift rate, in particular of its sign, has leaded us to a novel method to read a *local* clock, called *ICR Improved Clock Reading* method, that improves the clock accuracy by reducing at least by half the error growth between adjustments after an initial constant time that for most systems consists of a few hours. This primitive is relevant not only for refining the optimality result, but for suggesting an unexplored venue that might lead to further enhancements.

The clock synchronization problem, that guarantees the synchronization of a *known set* of distributed clocks through periodic adjustments, has a prohibitive cost when used in very large and dynamic populations. In addition, in most Internet–based applications clients do not need to be synchronized to the reference time during their lifetime. For these reasons, we address here the *more general* problem of providing many infinite

* The majority of this work was performed at Bell-Labs, Murray Hill.

processes dispersed in a WAN with a *close approximation* of the reference time when requested by them. This problem has a lower complexity than the previous one and its solutions can be easily adapted to solve the clock synchronization problem if needed. We propose a novel distributed *time service* built on top of our ICR primitive and resilient to Byzantine failures. Its efficiency is achieved by using Quorum Systems, a well–known technique that improves the system performance (i.e. access cost per operation, load balancing and scalability [20]). Our time service can be used as a fundamental building block in a number of Internet–based applications (i.e. e–commerce, timestamping, access control and so on).

The rest of the paper is organized as follows: Section 2 compares our results with previous work. Section 3 presents and analyzes the ICR method, and discusses the optimality result for external clock synchronization. Section 4 presents a time service built on top of the ICR method and Section 5 concludes the paper.

2 Related work

Improving the time accuracy of a set of distributed clocks by exploiting information related to its hardware drift rate is not a novel idea: for instance Schossmaier et al.[11][10] and NTP [15] include drift information in their clock adjustments. Fetzer and Cristian show in [3] how to build clocks with bounded drift rate from components *off–the–shell* and achieve high accuracy by means of *drift synchronized clocks.* Their work is close to ours for analyzing the behavior of the hardware drift rate, but diverges for its *strong* assumptions (i.e. requiring hardware support, PPS signals). Casimiro et al. [7] address time accuracy by improving the *remote clock reading error*, in particular by keeping track of previous round–trip delays and by means of drift information. This method is compatible with our primitives for its different prospective, though it is not suitable for our time service since it is not scalable. Our work diverges from previous solutions for reducing the maximum error growth associate to a clock between two consecutive adjustments. This novel prospective has led us to an improvement of the Cristian Fetzer optimality result for external clock synchronization.

Since the problem of providing many infinite processes with a *close approximation* of the reference time is not known in the literature, we compare an *adaptation* of our protocol that guarantees clock synchronization, to previous fault–tolerant clock synchronization protocols with clocks subject to drift. Previous solutions show a trade–off between efficiency, scalability in a wide area network and strong fault model abstraction. Protocols resilient to Byzantine failures such as [17][16][1][13] periodically perform estimations of the other clocks and therefore lack scalability and efficiency for large universe. In [5] Azevedo et al. synchronize clocks from a subset of time servers but rely on a *multi–step interactive* convergence approach to guarantee the validity of the adjustments. Notice this is not an issue when using Byzantine Quorum Systems since data consistency is guaranteed by the quorum construction [20]. On the other hand, solutions based on a hierarchical network structure and therefore highly scalable and suitable for the WAN, assume a weaker abstract fault model. Among these, the most widely spread Internet system is NTP [14][15]. CesiumSpray [8] is another solution for large–scale fault–tolerant systems based on a two–level hierarchy for a better scalability. Also in this case the scalability of the system and its performance degrade as the number of arbitrary failures increases. Our solution is novel for addressing at the same time efficiency, scalability and Byzantine failures: it provides a compromise between a strong failure model and system performance in a WAN by using Byzantine Quorum Systems. In case of high volume of client requests, as in the case of a time service, our approach is preferable to [1], employing $2b+1$ servers endowed with *GPS receivers* to tolerate b arbitrary failures, for improving the system load balance.

3 The ICR method

3.1 Preliminaries

It is well–known that an hardware clock function $H : \mathcal{RT} \rightarrow \mathcal{CT}$ is a monotonically increasing function mapping real time values $\mathcal{RT}$ into clock values $\mathcal{CT}$. A clock H with maximum drift rate ρ and granularity g is correct if $\forall t_1, t_2 : t_2 > t_1$ it measures the passage of time during the real time interval $[t_1, t_2]$ with an error of at most $\rho(t_2 - t_1) + g$

$$(1+\rho)^{-1}(t_2 - t_1) - g \leq H(t_2) - H(t_1) \leq (1+\rho)(t_2 - t_1) + g \tag{1}$$

Since for most quartz clocks ρ is of the order of 10^{-5} to 10^{-7}, we equate $(1+\rho)^{-1}$ with $1-\rho$ and $(1-\rho)^{-1}$ with $1+\rho$ and neglect the granularity g for the time intervals considered. Notice that the hardware clock function can be written in terms of its hardware drift rate δ_H as $H(t) := (1 + \delta_H(t))t$ with $\delta_H : \mathcal{RT} \rightarrow [-\rho, +\rho]$ and such that $\delta_H(t)$ is the drift rate of H at real time t. From now on we refer to $\delta_H(t)$ as δ_t. We consider a process p endowed with an hardware clock H and with a *virtual clock* C adjusted every Γ time units to a reference source (i.e. at real time $t_0, t_1, \ldots, t_i, \ldots$). We assume that $\forall i \geq 0$ process p receives at time t_i input $< T_i, \varepsilon_i >$ with T_i approximation of the real time t_i and ε_i its *inherited error*. In general ε_i is equal to the sum of the remote clock reading error and the error relative to the reference source. At time t_i the process synchronizes its virtual clock to T_i by resetting its offset γ_i. Process p maintains the following functions:

- a *virtual clock function* $C : \mathcal{RT} \rightarrow \mathcal{CT}$ mapping real time into clock values and such that $C(t) := \sum_{i \geq 0} C_i(t)$ with $C_i(t)$ defined as following:

$$C_i(t) := \begin{cases} H(t) + \gamma_i & \text{for } t \in [t_i, t_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$

  with $\gamma_i := T_i - H(t_i)$;
- an *error bound function* $E : \mathcal{RT} \rightarrow \Re$ mapping real time values into reals and such that $E(t) := \sum_{i \geq 0} E_i(t)$ with $E_i(t)$ defined as follows:

$$E_i(t) := \begin{cases} \varepsilon_i + (\rho + \rho^2)(H(t) - H(t_i)) & \text{for } t \in [t_i, t_{i+1}) \\ 0 & \text{otherwise} \end{cases}$$

Notice that $E_i(t)$ contains term $\rho^2(H(t) - H(t_i))$ because in some case this term might not be negligible (i.e. using Quorum Systems a process might receive an update after a long time). For simplicity, since this term is negligible in this section as it is less than $\rho^2\Gamma$, we drop it.

A *clock function* is a function mapping real time values to clock values. A *time interval* associated to a clock function D at real time t is defined as $I_D(t) := [D(t) - E(t), D(t) + E(t)]$ and is *correct* if it contains t. We assume in this section a *correct synchronization* at time t_i, that is $t_i \in [T_i - \varepsilon_i, T_i + \varepsilon_i]$. $ICR(t)$ is the time interval of real time t returned by the method. Therefore, the target of the ICR method is to return at any real time t a *correct* time interval of t with minimal size. In fact, by reducing the size of the time interval we improve the accuracy of its associate time value, that is the midpoint of that interval.

3.2 Method overview

Our ICR method aims at reducing the maximum error growth between adjustments without additional inputs. In this section we show that increasing the inherited error ε by a quantity proportional to its maximum drift rate, such as $\rho\Delta t$, leads to a conservative approach since it does not take into account any drift information. In fact, data regarding its hardware drift sign would lead to a refinement of the time interval: for instance, if a clock D proceeds slower than the real time then $[D(t), D(t) + E(t)]$ is a correct time interval of t and is more accurate than $I_D(t)$. In addition, the composition of two virtual clocks, one proceeding faster than the reference time and the other slower, would result in a more precise clock. These are some of the intuitions that has driven us to study the hardware drift sign and its variation between clock adjustments, elements that are at the base of our method. The ICR method can be summarized in two basic *actions*:

1. If ICR(t) detects no variation in the drift sign since its last clock adjustment, it computes a *special* clock function F, called *associate clock function*, with opposite drift sign with respect to δ_t and returns the interval $I_C(t) \cap I_F(t)$.
2. If the sign of the hardware drift has changed or the process is not able to detect it, a refinement of $E(t)$ is computed by replacing ρ with an approximation of $\max\{|\delta_t|, |\delta_{t_i}|\}$ (see Figure 2).

Before describing in detail the ICR method, we need to define the *desired* properties for the associate clock function F that can lead to optimal accuracy. We define below a class of clock functions $\mathcal{A}_{C,t}$ called *Associate Clock Functions* of $C(t)$. Notice that because of the periodic clock adjustments, for any real time t there exists an interval $[t_i, t_{i+1})$ that contains t.

Definition 1 *$F \in \mathcal{A}_{C,t}$ with $t \in [t_i, t_{i+1})$ if satisfies the following properties:*

1. *F is a clock function defined in $[t_i, t_{i+1})$ with ρ valid upper bound for its drift rate at t_i and t;*

2. *F and C are adjusted to the same clock value at time t_i, that is $F(t_i) = C(t_i)$;*
3. $|F(t) - C(t)| \geq \rho(H(t) - H(t_i))$.

Notice that Property 3 bounds by $\varepsilon + \rho(H(t) - H(t_i))$ the size of the time interval resulting from the intersection of $I_C(t)$ and $I_F(t)$.

Three issues arise from the high level description of ICR(t): (1) devise a test to detect the hardware drift sign at time t_i and t; (2) compute an associate clock function for $C(t)$; (3) compute a close bound for $\max\{|\delta_t|, |\delta_{t_i}|\}$. The first issue can be solved simply by comparing the hardware clock value with the left and right side of the current time interval. The accuracy of this test depends on $\rho(t - t_i)$ and ε_i, inherited error at the i^{th} adjustment, and improves as ρt_i becomes significant. Therefore, despite its simplicity, this test is able to detect drifts smaller than $\frac{\rho}{2}$ after $2\varepsilon_i\rho^{-1}$ time units, that is less than 67 minutes for $\rho = 10^{-5}$ and $\varepsilon_i = 20$ msec. The second issue is the most critical for the method. If the hardware drift sign is unchanged at time t and t_i, we compute an associate clock function of $C(t)$, denoted by $A_C(t)$ and defined in Lemma 2, such that the error growth of the resulting time value is reduced exactly by half. The third issue is solved by exploiting properties of the drift rate and of the time intervals.

In the following section we prove the correctness of the time interval associated to $A_C(t)$, analyze the maximum external deviation of ICR(t) and verify the correctness of the bound for $\max\{|\delta_{t_i}|, |\delta_t|\}$ at Figure 2 line (1). Figures 1 and 2 present the pseudo-code of ICR(t) for $t \in [t_i, t_{i+1})$. Notice that to leverage the notation we drop the subscription i in T_i, ε_i and γ_i. Therefore $< T, \varepsilon >$ represents the input received at time t_i from a more accurate reference source and *approx*() the function responsible for refining the error bound associated to $C(t)$ in case the drift sign has changed or the test cannot detect it due to its approximation. This is obtained by replacing ρ with a bound for $\max\{|\delta_{t_i}|, |\delta_t|\}$, Figure 2 at line (1). Notice that ICR(t) returns the intersection interval $I_C(t) \cap I_{A_C}(t)$ only if the hardware drift sign at time t_i and t is unchanged. In particular, Figure 1 at line (5) occurs when the hardware clock at time t_i and t is at the left side of its time interval I_C, and line (10) occurs if it is at the right side.

ICR(t):

1) if $H(t_i) \in I_C(t_i)$
2)   return *approx*()
3) else if $H(t_i) < T - \varepsilon$
4)   if $H(t) < C(t) - E(t)$
5)   return $I_C(t) \cap I_{A_C}(t)$
6)   else
7)   return *approx*()
8) else if $H(t_i) > T + \varepsilon$
9)   if $H(t) > C(t) + E(t)$
10)   return $I_C(t) \cap I_{A_C}(t)$
11)   else
12)   return *approx*()

Fig. 1. The ICR method.

approx():

1) $d \leftarrow \min\left\{\rho,\ \max\left\{\frac{|-\gamma|+E(t)}{C(t)-E(t)}, \frac{|-\gamma|+\varepsilon}{T-\varepsilon}\right\}\right\}$
2) $err \leftarrow \varepsilon + d(H(t) - H(t_i))$
3) return $[C(t) - err, C(t) + err]$

Fig. 2. The approx function.

3.3 Analysis of the method

In this section we show that the time interval returned by ICR is correct and compute an upper bound for its size. The correctness of time interval $I_C(t)$ follows from the drift bound ρ, condition (1) at Section 3.1.

Lemma 1. *If ρ is a valid upper bound of the hardware drift rate and the clock adjustments are correct, then $I_C(t)$ is a correct time interval of t.*

The following Lemma 2 computes an auxiliary clock function of $C(t)$ provided the same hardware drift sign at time t_i and t. Lemma 3 proves the correctness of ICR(t) at lines (4)-(5) and (9)-(10), and computes the maximum error for $ICR(t)$ in that case.

Lemma 2. *Let suppose the hardware drift sign be unchanged at real time t_i and t and $A_C(t)$ defined as follows:*

$$A_C(t) := \begin{cases} (1+\rho)H(t) + \gamma - \rho H(t_i) \text{ for } \delta_{t_i}, \delta_t \leq 0 \\ (1-\rho)H(t) + \gamma + \rho H(t_i) \text{ for } \delta_{t_i}, \delta_t \geq 0 \end{cases}$$

then $A_C(t) \in \mathcal{A}_{C,t}$.

Proof. Let suppose $\delta_{t_i}, \delta_t \geq 0$. Since the drift rate of $A_C(t)$ is equal to $\delta_t - \rho - \rho\delta_t$ and $\delta_t - \rho\delta_t > 0$, ρ is a valid bound for the drift rate of $A_C(t)$ at real time t_i and t. Therefore $A_C(t) \in \mathcal{A}_{C,t}$ since $C(t_i) = A_C(t_i)$ and $C(t) - A_C(t) = \rho(H(t) - H(t_i))$. Similarly if $\delta_{t_i}, \delta_t \leq 0$.

Lemma 3. *If $H(t_i) \notin I_C(t_i)$ and $H(t) \notin I_C(t)$, $ICR(t)$ returns a correct time interval with an error bound equal to $\varepsilon + \frac{\rho}{2}(H(t) - H(t_i))$.*

Proof. Let suppose $H(t_i) > T + \varepsilon$ and $H(t) > C(t) + E(t)$. Since $I_C(t)$ is a correct time interval, then $\delta_{t_i}, \delta_t > 0$. Therefore, by Lemma 2 and Lemma 1, $I_C(t) \cap I_{A_C}(t)$ is a correct time interval and its size is equal to $2\varepsilon + \rho(H(t) - H(t_i))$.

The following Lemma 4 proves the correctness of the test performed by ICR(t) at lines (3)-(4) and (8)-(9). It reduces to five the number of feasible cases based on the evaluation $H(t_i) \in I_C(t_i)$ and $H(t) \in I_C(t)$. Lemma 5 proves the correctness of the time interval returned by *approx*(), invoked by ICR(t) at lines (7) and (12). Theorem 1 states our main result. Due to space restrictions the proofs are omitted.

Lemma 4. *If $H(t_i) \in I_C(t_i)$ then $H(t) \in I_C(t)$. Otherwise, if $H(t_i) < T - \varepsilon$ then $H(t) < C(t)$ and if $H(t_i) > T + \varepsilon$ then $H(t) > C(t)$.*

Lemma 5. *If $t \in [t_i, t_{i+1})$ and $err = \varepsilon + \max\{|\delta_{t_i}|, |\delta_t|\}(H(t) - H(t_i))$, then $[C(t) - err, C(t) + err]$ is a correct time interval at t. If $H(t) \in I_C(t)$ then*

$$\max\{|\delta_{t_i}|, |\delta_t|\} \leq \min\left\{\rho\ ,\ \max\left\{\frac{|-\gamma| + E(t)}{C(t) - E(t)}, \frac{|-\gamma| + \varepsilon}{T - \varepsilon}\right\}\right\}$$

Theorem 1. *ICR(t) returns a correct time interval and its maximum error growth is at most $\frac{\rho}{2}\Gamma$ after $4\varepsilon\rho^{-1} + 6\Gamma + 2\varepsilon$ time units if the process is adjusted correctly every Γ time units.*

3.4 Optimality result

In [1] Fetzer and Cristian prove that the best maximum external deviation achievable by a clock synchronized at least every Γ time units is equal to $\Delta + \Lambda + \rho\Gamma$ with Λ the remote reading error and Δ the error relative to the reference source. We improve this result by showing a method, the ICR method, that under the *same assumptions* reduces the maximum external deviation to $\Delta + \Lambda + \frac{\rho}{2}\Gamma$ after $4\varepsilon\rho^{-1} + 6\Gamma + 2\varepsilon$ time units (see Theorem 1). Notice that Fetzer and Cristian assume in [1] that hardware clocks are within a linear envelope of the real time but do not exploit this assumption in their proof that is based on an *adversary argument* (i.e. the adversary can set $\delta_{t_i} = -\rho$ and $\delta_t = \rho$).

It is important to notice that our ICR method do not assume a specific hardware drift distribution (i.e. its drift changes slowly between two adjustments) but only a valid drift bound and a correct periodic adjustment. This feature makes our primitive general and therefore widely applicable. In addition, for most systems $4\varepsilon\rho^{-1} + 6\Gamma + 2\varepsilon$ consists of several hours: for instance if $\rho = 10^{-4}$, $\Gamma = 30$ minutes and $\varepsilon = 20$ msec the initial time is approximately equal to 3 hours 15 minutes and its improvement $\frac{\rho}{2}\Gamma$ consists of at least 90 msec.

4 An ICR–based time service

In this section we address the problem of providing many infinite processes with a close approximation of the reference time when requested. This problem is *more general* than the clock synchronization problem and suitable for very large and dynamic populations. Our target is to design a *distributed time service*, called **time**(), that is invoked by a generic process in a WAN and with minimum requirements, and such that the *external deviation* of the value returned by **time**(), its distance from the real time, is bounded by a *a priori known* constant. In this section we propose the first implementation of a distributed time service; it is resilient to malicious failures and based on Quorum Systems.

4.1 System Model

We consider an asynchronous distributed system consisting of a set $\mathcal{U} = \{S_1, \ldots, S_n\}$ of time servers of size n, an unknown and dynamic set of clients and a specialized process called the *GPS source*. Processes communicate by message passing and have a unique identification number. Time servers can exhibit up to $b < \frac{n}{4}$ Byzantine server failures, clients can be transient and might behave maliciously, and the GPS source can crash. All process communications are done via quorums, by using a b–Masking Quorum System $\mathcal{Q}$: a collection of subsets of $\mathcal{U}$ such that any two subsets, quorums, overlap in at least $2b+1$ servers (consistency property), and for any failure configuration there is a quorum consisting of correct servers (availability property), see [20] for details.

GPS source In [1] Fetzer and Cristian show that $2b+1$ time servers provided with *time receivers* are sufficient to guarantee external clock synchronization when b servers are arbitrary faulty. The approach proposed in this section releases servers from being attached to the *time receivers* as it involves only a specialized process, called *GPS source*, that periodically synchronizes the time servers. This approach shows a trade–off between precision on one side and scalability, reconfigurability and costs on the other. Our architecture is preferable over [1] for applications in which *flexibility* in the configuration and *reduced costs* of the hardware components needed to access external reference time signal play an important role. On the other hand, this approach increases the error bound since introduces an additional remote clock reading error. In this section we discuss this novel architecture since adapting [1] to our time service is straightforward.

Several issues related to a single point of failure arise when implementing a fault–tolerant GPS source: the system has to be resilient to benign failures of the GPS source, and to transient omissions of the GPS receivers (i.e. see [19]) and no other process should be able to synchronize the time servers or corrupt the *GPS update*. In our model the GPS source is a *specialized* and *trusted party*: servers verify the identity of the sender by using authentication schemes based on symmetric or public keys cryptographic systems and also verify the integrity of the transmitted update. The GPS source copes with occasional discontinuities of the GPS receiver by using multiple *GPS antennas* pointed in different directions and by averaging their time values giving greater weights to those with more satellites, see Figure 3 line (5). Therefore, in our system model a *GPS source* is a specialized and trusted process connected to a set of *GPS antennas* that every Γ time units diffuses a time interval with maximum deviation Δ to a *quorum* of servers.

Time servers A time server is *correct* if it follows the specifications. Each time server is endowed with an hardware clock H and a virtual clock C that is adjusted by the GPS source. We adopt, for its simplicity and efficiency, Cristian probabilistic reading method [2] to estimate a remote clock. It is based on the round trip delay elapsed between the sending of the time request and the arrival of the reply. Therefore, if m is the minimum message delay and M a maximum acceptable round trip delay, the maximum acceptable *remote clock reading error* Λ_c is defined as following: $\Lambda_c = \frac{M(1+\rho)^2 - 2m}{2}$.
Function $Exp(I)$ approximates the *remote time interval* I to the real time the process receives it. It is defined as $Exp(I) := [\min(I) + m(1-\rho),\ \max(I) + (1+\rho)^2(D-m)]$ where $D \leq M$ is the computed round trip message delay. The maximum *remote interval reading error* is $\Lambda = \Lambda_c + \rho\Lambda_c$, but for simplicity we drop $\rho\Lambda_c$.

4.2 The protocol

In this subsection we propose an implementation of a time service that is always correct and alive in condition of network stability. Notice that the time servers return $ICR(t)$ instead of the time interval $I_C(t)$ for a better accuracy. We describe first the GPS synchronization in Figures 3 and 4, and then the client–side in Figure 5, and denote a remote call performed on a quorum Q to request $p(..)$ by Q-RPC(Q, $p(..)$).

The GPS source is responsible for diffusing a correct reference time to a quorum of servers every Γ time units. The GPS source runs a two–phase protocol: in the first phase it notifies the upcoming synchronization to a quorum of servers to allow servers to compute the round trip delay, and in the second phase it sends the average time interval computed based on data received from its different GPS antennas, Figure 3 lines (5)-(6). Servers adjust their virtual clock to the midpoint of the *expanded GPS interval* and reset their inherited error, Figure 4 at lines (4)-(6). Notice that because of the probabilistic reading method, in order to guarantee the periodic *GPS adjustment* of a full quorum set, the GPS source needs a positive acknowledgment from at least $|Q| - b$ servers.

GPS():

for every Γ time
  do
    choose $Q \in \mathcal{Q}$
    $\{ack_i\}_{i \in Q} \leftarrow$Q-RPC($Q$,start($id$))
    $I \leftarrow$Aver-GPS()
    $\{ack_i\}_{i \in Q} \leftarrow$Q-RPC($Q$,adj($I, id$))
  while $|\{ack_i\}_{i \in Q}| < |Q| - b$
end for

Fig. 3. The GPS side.

start (id)

auth (id)	auth GPS
$R \leftarrow H(now)$	remote reading
return ack	

adj (I, id)

if $(H(now) - R)(1 + \rho) > M$	remote reading
return $nack$	
auth(id)	auth GPS, data integrity
$T \leftarrow mid(Exp(I))$	
$\gamma \leftarrow T - H(now)$	offset adjust
$\varepsilon \leftarrow \frac{\lvert Exp(I) \rvert}{2}$	error adjust
return ack	

Fig. 4. The server side.

In order to bound the external deviation of the time value returned by **time**(), it is important to detect any GPS crash. Notice that servers cannot establish, based on the messages received, if a GPS source has crashed because they might not receive a GPS update for a long time due to the Quorum System approach. Therefore, to avoid additional communications, we rely on correct clients to detect GPS failures. Each client verifies if the size of the time interval computed exceeds the maximum size MAX allowed by the protocol, Figure 5 line (6), and if this is the case notifies it to a quorum of servers, Figure 5 line (9). Upon receiving this notification each server verifies that the GPS source is down, and if it does not receive a response within x time units, sends a request to a back–up GPS source. The back–up GPS source becomes *active* only after receiving $b + 1$ identical requests from different servers. In this way neither a faulty client, nor a faulty server can simulate a GPS crash. This solution does not rule out the possibility that two GPS sources might be active at the same time but since this does not affect the correctness of the protocol, that is the main focus of our discussion, we keep this issue aside.

Clients simply pool time information from a quorum of servers, this makes the system resilient to malicious clients. Each client requests the current time intervals and derives from $\{Exp(ICR_i)\}_{i \in Q}$ a fault–tolerant average interval by applying Marzullo algorithm [4], Figure 5 lines (3)-(5). The client returns the midpoint of the average interval if it does not exceed the maximum size MAX, otherwise sends a notification to a quorum of servers, waits for $x + 2M$ time units, lines (9)-(10), and tries again.

time():

1) while ($true$)
2) $T \leftarrow H(now)$
3) $\{ICR_i\}_{i \in Q} \leftarrow$ Q-RPC(Q, ICR())
4)   if $(H(now) - T)(1 - \rho) \leq M$
5) $A \leftarrow Aver(\{Exp(ICR_i)\}_{i \in Q})$
6)   if $|A| \leq MAX$
7)   return mid(A)
8)   else
9) $\{ack_i\}_{i \in Q} \leftarrow$ Q-RPC(Q, notify())
10)   wait($x + 2M$)

Fig. 5. The client side of time.

Lemma 6. *If ρ is a valid upper bound of the hardware drift rate, then* **time**() *implements a correct time service and it is always available in condition of network stability.*

5 Conclusion

The contribution of this paper is twofold and has both theoretical and practical interest. We have proposed a general primitive to read a local clock, that improves the Fetzer and Cristian optimality result for external clock synchronization and opens a venue for further improvements.

We have also analyzed a problem more general than the clock synchronization problem and suitable for large and dynamic populations, and proposed an efficient time service based on our ICR primitive, resilient to malicious failures and whose applicability is large–scale.

References

1. C. Fetzer and F.Cristian. *Integrating external and internal clock synchronization.* Journal of Real-Time Systems, 12(2), pp. 123-171, March 1997.
2. F.Cristian. *Probabilistic clock synchronization.* Distributed Computing 3(3), pp. 146-158, 1989.
3. C.Fetzer, F.Cristian. *Building fault-tolerant hardware clocks.* In Proceedings of the 7th IFIP International Working Conference on Dependable Computing for Critical Applications, pp. 59-78, Jan 1999.
4. K.Marzullo. *Tolerating Failures of Continuous-Valued Sensors.* ACM Transactions on Computer Systems, 8(4), pp. 284-304, Nov 1990.
5. M.Moraes de Azevedo, D.M.Blough. *Multi-step Interactive Convergence:An Efficient Approach to the Fault–Tolerant Clock Synchronization of Large Multicomputers.* IEEE Transactions on Parallel and Distributed Systems, 9(12), Dec 1998.
6. K.Marzullo, S.Owicki. *Maintaining the Time in a Distributed System.* In Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing (PODC '83) pp. 295-305, Montreal, Aug 1983.
7. A. Casimiro, P. Martins, P. Verissimo, L. Rodrigues. *Measuring Distributed Durations with Stable Errors.* Proceedings of the 22nd IEEE Real-Time Systems Symposium, London, Dec 2001.
8. P. Verissimo, L. Rodrigues, A.Casimiro. *Cesiumspray: a precise and accurate global clock service for large-scale systems.* Real-Time Systems, 12(3), pp. 243-294, May 1997.
9. U.Schmid. *Orthogonal Accuracy Clock Synchronization.* Chicago Journal of Theoretical Computer Science, pp. 3-77, Aug 2000.
10. K.Schossmaier, B.Weiss. *An Algorithm for Fault–Tolerant Clock State&Rate Synchronization.* Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS '99), Lausanne, pp. 36-47, Oct 1999.
11. K. Schossmaier. *An Interval-based Framework for Clock Rate Synchronization.* Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC '97), pp. 169-178, Santa Barbara, Aug 1997.
12. U.Schmid, K.Schossmaier. *Interval-based Clock Synchronization*, Journal of Real-Time Systems 12(2), pp. 173-228, Mar 1997.
13. B.Barak, S.Halevi, A.Herzberg, D.Naor. *Clock synchronization with faults and recoveries.* Proceedings of the 19th annual ACM symposium on Principles of Distributed Computing (PODC '00), pp. 133-142, Portland, July 2000.
14. D.L.Mills. *On the accuracy and stability of clocks synchronized by the Network Time Protocol in the Internet system.* ACM Computer Communication Review 20(1), pp. 65-75, Jan 1990.
15. D.L.Mills. *Adaptive hybrid clock discipline algorithm for the Network Time Protocol.* IEEE/ACM Trans. Networking 6(5), pp. 505-514, Oct 1998.
16. D.Dolev, J.Y.Halpern,B.Simons, R.Strong. *Dynamic fault–tolerant clock synchronization.* JACM 42(1), pp. 143-185, Jan 1995.
17. J.Lundelius–Welch, N.Lynch. *A new fault–tolerant algorithm for clock synchronization.* Information and Computation, 77(1), pp. 1-36, Jan 1988.
18. *The Science of Time-keeping.* Hewlett Packard, application Note 1289.
19. D. Höchtl, U.Schmid. *Long–Term Evaluation of GPS Timing Receivers Failures.* Proceedings of the 29^{th} Precise Time and Time Interval (PTTI) Systems and Applications Meeting, Long Beach, Dec 1997.
20. D.Malkhi, M.Reiter, A.Wool *The load and availability of Byzantine Quorum Systems.* SIAM J.Computing 29(6), pp. 1889-1906, (2000).