A PROBABILISTIC APPROACH TO DISTRIBUTED CLOCK SYNCHRONIZATION

Flaviu Cristian
IBM Almaden Research Center
San Jose, Ca 95120-6099

ABSTRACT: A probabilistic method is proposed for reading remote clocks in distributed systems subject to unbounded random communication delays. The method can achieve clock synchronization precisions superior to those attainable by previously published clock synchronization algorithms. Its use is illustrated by presenting a time service which maintains externally (and hence, internally) synchronized clocks in the presence of process, communication and clock failures.

INTRODUCTION

In a distributed system, *external* clock synchronization consists of maintaining processor clocks within some given maximum deviation from a time reference external to the system. *Internal* clock synchronization keeps processor clocks within some maximum relative deviation of each other. Externally synchronized clocks are also internally synchronized. The converse is not true: as time passes internally synchronized clocks can drift arbitrarily far from external time.

Clock synchronization is needed in many distributed systems. Internal clock synchronization enables one to measure the duration of distributed activities that start on one processor and terminate on another processor and to totally order distributed events in a manner that closely approximates their real time precedence. To allow exchange of information about the timing of events with other systems and users, many systems require external clock synchronization. For example external time can be used to record the occurrence of events for later analysis by humans, to instruct a system to take certain actions when certain specified (external) time deadlines occur, and to order the occurrence of related events observed by distinct systems.

This paper proposes a new approach for reading remote clocks in networks subject to unbounded random message delays. The method can be used to improve the precision of both internal and external synchronization algorithms. Our approach is *probabilistic* because it does not guarantee that a processor can always read a remote clock with an a priori specified precision (such a guarantee cannot be provided when there is no bound on message delays). However, by retrying a sufficient number of times, a process can read the clock of another process with a given precision with a probability as close to one as desired. An important characteristic of our method is that when a process succeeds in reading a remote clock, it *knows* the actual reading precision achieved.

The use of the remote clock reading method is illustrated by describing a distributed time service which maintains externally synchronized clocks despite process, communication and clock failures. The service is implemented by a group of time servers which execute a simple probabilistic clock synchronization protocol. After presenting the protocol and its performance, we conclude by comparing it with other published clock synchronization protocols.

MESSAGE DELAYS

To synchronize the clocks of their host processors, time server processes communicate among themselves by sending messages via a communication network. Since there is a one to one correspondence between time server processes and processors, we do not distinguish between processes and processors. For example, when we say "the clock of process P" we mean "the clock of the processor on which P runs".

In distributed systems the task of synchronizing clocks is made difficult (among other things) by the existence of unpredictable communication delays. Between the moment a process P sends a message to a process Q and the moment Q receives the message, there is an arbitrary, *random* real time delay. A minimum min for this delay exists. It can be computed by counting the time needed to prepare, transmit, and receive an empty message in the absence of transmission errors and any other system load. In general, one does not know an upper bound on message transmission delays. These depend on the amount of communication and computation going on in parallel in the system, on the possibility that transmission errors cause messages to be retransmitted several times, and on other random events, such as page faults, process switches, the establishment of new communication routes, or a freeze of the activity of a process caused by a human operator who pushes the 'halt' button on the panel of the processor hosting that process.

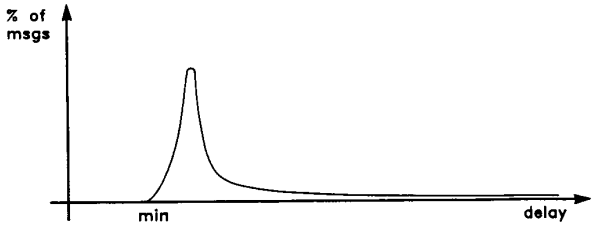

Figure 1.

Measurements of process to process message delays in existing systems indicate that typically their distribution has a shape resembling that illustrated in Figure 1. This distribution has a maximum density at a mode point between the minimum delay min and the median delay, usually close to min, with a long thin tail to the right. For instance, a sample measurement of 5000 message round trip delays between two light-weight MVS processes (running on two IBM 4381 processors connected via a channel-to-channel local area network) performed at the Almaden Research Center [D], indicates a median round trip delay of 4.48 milliseconds situated between a minimum delay of 4.22 milliseconds and an average delay of 4.91 milliseconds. While the maximum observed delay in this experiment (during which no route changes or 'halt' button pushes occurred) was very far at the right: 93.17 milliseconds, 95% of all observed delays were shorter than 5.2 milliseconds.

PREVIOUS WORK

Most published clock synchronization algorithms (e.g. [CAS], [DHSS], [L], [LM], [LWL], [S], [ST]) assume the existence of an upper bound max on real time message transmission delays. If the delays experienced by delivered messages are smaller than max with probability 1, these algorithms keep clocks within a maximum

CH2706-0/89/0000/0288$01.00 © 1989 IEEE

relative deviation greater than max-min with probability one. It is known [LL] that the closeness with which clocks can be synchronized with certainty (i.e. with probability one) is limited: n clocks cannot be synchronized with certainty closer than (max-min)(1-1/n), even in the absence of failures and drift.

Other authors (e.g. [GZ], [M]) adopt the premise that message delays are unbounded, and use as upper bounds on synchronization message delays the timeout delays employed for detecting communication failures between processes. Such timeouts are introduced by system designers to prevent situations in which some process P waits forever for a message from another process Q that will never arrive (for example because of a failure of Q). Since message delays are unbounded, it is understood that a small percentage of messages may need more than a given timeout delay to travel between processes, i.e. there is a chance that "false" communication failures are detected. This is the price paid for letting systems subject to unbounded message transmission delays continue to work despite process failures and message losses. To reduce the likelihood of "false" alarms, a timeout delay is conservatively estimated from network delay statistics to ensure that message delays are smaller than the chosen timeout with a very high probability p (typically p>0.99). If such a timeout delay is denoted by "maxp", the best synchronization precision achievable by the algorithms proposed in [GZ],[M] can be characterized as being 4(maxp-min).

ASSUMPTIONS ON CLOCKS, PROCESSES, AND COMMUNICATION

Each time server process has access to the hardware clock H of its host processor. To simplify our presentation, we assume these clocks have a much higher resolution than the time intervals (e.g. process to process communication delays) which must be measured. For example, if the delays observable are of the order of milliseconds, we assume the hardware clocks have a microsecond resolution. A clock H is *correct* if it measures the length t'-t of any real time interval [t,t'] with an error of *at most* $\rho(t-t')$, where ρ is the maximum clock drift rate from external (or real) time specified by the clock manufacturer:

(C) $\quad (1-\rho)(t-t') \leq H(t)-H(t') \leq (1+\rho)(t-t')$.

In the above formula, it is implicit that the delay t-t' is long enough so that the worst case error in measuring its length caused by the discrete clock granularity is negligible compared to that due to drift. For most types of quartz clocks, the constant ρ is of the order of 10^{-5}. For example the worst actual drift rate measured for the microsecond resolution clocks existing on the IBM 4381 processors in our laboratory is $6*10^{-6}$ [D]. Since ρ is such a small quantity, we ignore in this paper terms of the order of ρ^2 or smaller (e.g. we equate $(1+\rho)^{-1}$ with $(1-\rho)$ and $(1-\rho)^{-1}$ with $(1+\rho)$). A clock *failure* occurs if the clock correctness condition (C) is violated. Examples of clock failure types are: crash failures (i.e. the clock stops), timing failures (e.g. a change in the frequency of the quartz oscillator driving the clock counter causes the clock value to be incremented too fast or too slowly), and Byzantine failures (e.g. the clock counter displays a nonmonotonic time because some of its bits are stuck at 0 or 1). To simplify our presentation, we assume initially that processor clocks are correct. We relax this assumption later, by showing how one can detect and handle arbitrary clock failures.

We assume that message delays between processes are *unbounded*. As we will see later, the closer the distribution of such delays resembles that of Figure 1 (i.e. the closer the median delay is to min), the better our probabilistic clock synchronization algorithms perform. What is remarkable, however, is that their correctness *does not* depend on any assumption about the particular shape of the message delay density function. We also assume that, to let processes continue to work despite process failures or message losses, a timeout delay maxp is chosen. The adoption of such a timeout delay divides observable network behaviors into two classes. A communication path (P,Q) between processes P and Q is said to function *correctly* if any message sent by P is delivered uncorrupted to Q within maxp time units. If a message accepted at one path end is never delivered at the other end or is delivered after more than maxp time units, the path suffers a late timing or *performance failure* [CASD]. We assume that communication channels between processes can only be affected by performance failures.

Processes undergo state transitions in response to message arrivals and timeout events generated by timers. To simplify our presentation we assume that between the occurrence of a timeout and the invocation of the associated timeout handler there is a null (process scheduling) delay and that process timers advance at the same rate as the clocks of the underlying processors. Thus, a correct process which at real time t sets a timer to measure W time units, is awakened in the real time interval $[t+(1-\rho)W, t+(1+\rho)W]$. We say that a process behaves *correctly* if in response to trigger events (such as message arrivals or timeout occurrences) it behaves in the manner specified. The specification prescribes the state transitions which should occur and the time intervals within which these transitions should occur. If, in response to some trigger event, a process never performs its specified state transition or undergoes it too early or too late (i.e. outside the time interval specified), the process is said to suffer a *timing failure* [CASD]. Processes which crash, omit to receive or send certain messages, respond too slowly to trigger events (because of excessive load or slow timers), or time out too early (because of timers running at speeds greater than $1+\rho$) are examples of timing failures. We assume processes can suffer only *timing* failures.

ATTEMPTING TO READ A REMOTE CLOCK

To read the clock of a process Q, a process P sends a message ("time=?") to Q. When Q receives the message it replies with a message ("time=",T), where T is the time on Q's clock. If P does not receive a reply because of a failure, its attempt at reading Q's clock fails. Assume that P receives a reply and let D be half of the round trip delay measured on P' clock between the sending of the ("time=?") message and the reception of the ("time=",T) message.

Theorem: If the clocks of processes P and Q are correct, the value displayed by Q's clock when P receives the ("time=",T) message is in the interval $[T+\min(1-\rho), T+2D(1+2\rho)-\min(1+\rho)]$.

Proof: Let t be the real-time when P receives the ("time=",T) message from Q, and $C_Q(t)$ be the value displayed by Q's clock at that time. Let $\min+\alpha$, $\min+\beta$, $\alpha \geq 0$, $\beta \geq 0$, be the real time delays experienced by the ("time=?") and ("time=",T) messages, respectively, and let 2d be the real time round trip delay:

(1) $2d = 2\min + \alpha + \beta$.

Since α and β are positive, (1) implies:

(2) $0 \le \beta \le 2d - 2\min$.

From the definition of β, and the fact that Q's clock can run at any speed in the interval $[1-\rho, 1+\rho]$, we can infer that, at real time t, Q's clock satisfies the condition:

(3) $C_Q(t) \in [T+(\min+\beta)(1-\rho),\ T+(\min+\beta)(1+\rho)]$.

By combining (2) and (3) we obtain:

(4) $C_Q(t) \in [T+\min(1-\rho), T+(2d-\min)(1+\rho)]$.

Since the clock that P uses to measure the round trip delay can drift at a rate of at most ρ from real time, it follows that

(5) $d \le D(1+\rho)$.

By substituting (5) into (4) we get (after some simplifications):

(6) $C_Q(t) \in [T+\min(1-\rho),\ T+2D(1+2\rho)-\min(1+\rho)]$.

End of proof.

The above theorem indicates that P can determine an interval which contains Q's clock value if *it measures* the round trip delay 2D. Since possible scenarios such as $\alpha=2(d-\min)$, $\beta=0$ and $\alpha=0$, $\beta=2(d-\min)$ are indistinguishable to P, and we assume that P does not know the drift rate of Q's clock or its own clock, the value $C_Q(t)$ can be *any* point in this interval. In other words: $[T+\min(1-\rho), T+2D(1+2\rho)-\min(1+\rho)]$ is the *smallest* interval which P can determine in terms of T and D that covers Q's clock value.

Since P has no means of knowing exactly where Q's clock is in the interval (6), the best it can do is to *estimate* $C_Q(t)$ by a function $C_Q^*(T,D)$ of what *it knows*, that is, T and D. In doing so, the actual error that P makes is:

$$| C_Q^*(T,D) - C_Q(t) |.$$

P minimizes the maximum error it can make in estimating $C_Q(t)$ by choosing $C_Q^*(T,D)$ to be the *midpoint* of the interval (6):

(7) $C_Q^*(T,D) \equiv T + D(1+2\rho) - \min\rho$.

For this choice of $C_Q^*(T,D)$, the maximum error e that P can make when reading Q's clock is half the length of the interval (16):

(8) $e = D(1+2\rho) - \min$.

Any other estimate choice leads to a bigger maximum error. We refer to the expression (7) as "P's reading of Q's clock" and to (8) as "P's reading error" or "P's reading precision".

READING A REMOTE CLOCK WITH A SPECIFIED PRECISION

Formula (8) can be interpreted as follows: the shorter the round trip delay is, the smaller P's error in reading Q's clock is. Thus, if P wants to achieve a reading error smaller than a certain specified maximum error (or precision) ε, it must *discard* any reading attempt for which it measures an actual round trip delay greater than 2U, where

(9) $U = (1-2\rho)(\varepsilon+\min)$.

Indeed, by (8), such clock readings can lead to actual reading errors greater than ε. For this reason, we call a round trip delay smaller than 2U *successful*, and refer to 2U as the *timeout delay* necessary for achieving the reading precision ε. When the process P observes a successful round trip, we say that it *reaches rapport* with Q.

The closer U is to min, the better P's reading precision is. However, since in the worst case P's timer can run at a rate as fast as $1+\rho$, P must chose a timeout delay greater than

(10) $U_{min} = \min(1+\rho)$,

to ensure that between the sending of a message and its reception there is a real time delay of at least min. To achieve the best possible precision for which there exists a positive probability of rapport, P must chose a timeout delay as close to U_{min} as possible. For such a limit timeout delay, formula (8) implies that the best reading precision achievable by a clock reading experiment is

(11) $e_{min} = 3\rho\min$.

The first two ρs correspond to the relative drift between Q's clock and P's clock while the ("time=",T) message travels between Q and P, and the third ρ corresponds to P's error in setting its timeout delay so that it measures at least min real time units.

Let p be the probability that P observes a round trip delay greater than 2U. The larger U is, the smaller p will be. Conversely, the smaller U is, the larger p will be. Thus, there exists a *fundamental trade-off* between the precision achievable when attempting to read a remote clock and the probability 1-p of success. The better the desired precision is, the smaller is the probability of success. Conversely, the worse the precision is, the greater is the probability of success. In the limiting case, if a maximum real time message delay max is known, by settling for a remote clock reading precision of $\max(1+3\rho)$-min (corresponding to a timeout delay of $\max(1+\rho)$), one obtains a *deterministic* remote clock reading algorithm (similar to the ones used by the synchronization algorithms presented in [CAS], [DHSS], [L], [LM], [LWL], [S], [ST]) which *always* achieves rapport. The price for such a choice is poor precision.

Consider now a certain specified precision ε and the associated probability p that a reading attempt fails. For this precision, the probability that process P reaches rapport with process Q can be increased if *several* clock reading attempts are allowed before P gives up. To achieve a certain degree of independence between successive attempts, these should be separated by a minimum waiting delay W. This delay must be chosen so as to ensure that if P and Q stay connected and correct, then any transient network traffic bursts that may affect their communication disappear within W clock time units with high probability. (A solution to the problem of how to adapt to slower, nonbursty, network load changes is sketched later.) To avoid P attempting, ad infinitum, to read Q's clock when Q is permanently partitioned from P or has crashed, one must decide on a maximum value k for the number of successive attempts that P is allowed to make. For a given choice of k, allowing for up to k reading attempts increases the probability of success to $1-p^k$. Since $p<1$, this probability can be made arbitrarily close to 1 by choosing a sufficiently large k.

For large values of k and a choice of W that ensures independence between successive reading attempts, Bernoulli's law yields that the average number of reading attempts needed for achieving rapport is $(1-p)^{-1}$. Since each attempt costs two messages, it follows that the average number of messages $\overline{m}$ for achieving rapport is

$$(12) \qquad \overline{m} = \frac{2}{(1-p)}.$$

Formulae (8) and (12) indicate the existence of a *continuum* of different clock reading algorithms indexed by different timeout delays U: from aggressive but risky algorithms indexed by U's close to min which are capable of achieving high precisions by possibly using a very large number of messages, to low risk "deterministic" algorithms indexed by U's close to max which achieve poor precisions by using a small number of messages.

A DISTRIBUTED TIME SERVICE.

The probabilistic clock reading method described above can be used to improve the precision achievable by most of the internal clock synchronization algorithms surveyed in [S] by letting time servers read probabilistically the remote clock values used as inputs to the convergence functions mentioned there. Instead of exploring this avenue, we devote the rest of the paper to describing a simple distributed time service which provides external clock synchronization.

The goal is to keep clocks synchronized to an official source of external time signals, such as the Universal Time Coordinated (UTC) signals broadcast by the WWV radio station of NBS. Commercially available receivers (e.g. [K]) can receive such signals. The receivers can be attached to processors via dedicated busses. To guard against a physical receiver failure, it is possible to pair physically independent receivers into a *self-checking* receiver unit, by continuously comparing their results, and interpreting any disagreement among them as a failure of the pair [K]. If no multiple failures occur, a self-checking receiver either displays correctly the external time or signals an error. We assume that all radio receivers used by the time service are self-checking. We also assume that, for reasons of economy, only certain processors, called *masters*, have time receivers attached to them. The other processors are referred to as *slaves*. To simplify our presentation we initially assume the existence of a unique, continuously available, master time source. Issues related to the implementation of this master time source by a group of redundant physical masters are discussed later. To further simplify the presentation, we do not distinguish between real (or atomic) time and astronomical UTC time, that is, we ignore problems related to the existence of yearly UTC time discontinuities known as "leap seconds". (For a discussion of the differences between these two time references, see [KO].) We furthermore assume that the official source of external time is reliable and that its signals are always available for reception by the radio receivers attached to master processors. The investigation of the issues related to maintaining synchronization in the presence of erroneous external time signals or in the absence of such signals constitutes a research topic in its own right.

Continuously adjustable clocks

Some processor architectures enable the speed of a hardware clock to be changed by software while others do not. Since the former make clock management dependent on the particular commands available for changing clock speeds, in this paper we chose to discuss the latter alternative. To compensate for the fact that the speed of a hardware clock H is not adjustable, a logical clock C with adjustable speed is implemented in software. The value of C is defined as the sum of the local hardware clock H and a periodically computed adjustment function A:

$$C(t) \equiv H(t) + A(t).$$

To avoid logical clock discontinuities (i.e. jumps) A must be a continuous function of time. For simplicity we consider only linear adjustment functions

$$A(t) = m*H(t)+N,$$

where the m and N parameters are computed periodically as described below. If, at local rapport time L, a slave estimates that the master clock displays time M, $M \neq L$, the goal is to increase (if $M>L$) or decrease (if $M<L$) the speed of the slave clock C so that it will show time $M+\alpha$ (instead of $L+\alpha$) α clock time units after rapport, where α is a positive clock *amortization* parameter. Since at the beginning and end of the amortization period the slave clock displays the values $L=H(1+m)+N$ and $M+\alpha=(H+\alpha)(1+m)+N$, respectively, where H is the hardware clock value at rapport, by solving the above system of equations we conclude that the parameters m, N must be set to

$$(A) \qquad m = (M-L)/\alpha, \quad N = L - (1+m)*H$$

for the α clock time units following rapport. After the α amortization period elapses, at local time $L'=M+\alpha$, the slave clock C can be allowed to run again at the speed of the local hardware clock until the next rapport by setting m to 0 and (to ensure continuity of C) N to L'-H', where H' is the value displayed by the hardware clock at the end of the amortization period.

The Master-Slave Synchronization Protocol

The time service is implemented by a group of distributed time server processes, one per correctly functioning processor in the system. The master server running on the master processor M keeps the master logical clock C_M within a maximum deviation *em* (external-master) of external (or real) time. A slave server S keeps its logical clock C within a maximum deviation *ms* (master-slave) from the master clock. In this way the maximum deviation *es* of a slave from external time will be $em+ms$ and the maximum relative deviation of two slaves will be $ss=2ms$.

Since the protocol used for synchronizing a master clock to the clock of an attached self-checking receiver is similar to that used for synchronizing a slave clock to a master clock, we only describe the latter in detail. The main difference between the two protocols lies in the variability of observed round trip delays. While a variability of the order of milliseconds is reasonable for master slave communications, variabilities much smaller can be achieved for the communication between a master time server and the self-checking receiver attached via a dedicated bus (for instance by ensuring that the master server does not relinquish control of the master CPU during a receiver clock reading attempt). By formula (8) this yields a very high receiver clock reading precision. If this high reading precision is supplemented by the adoption of a high master clock resynchronization frequency, the *em* constant can be

kept so small that it is reasonable to assume in what follows that a master clock runs at the same speed as the external time.

The absence of master drift, the fact that for current local area network technology round trip delays smaller than 10 seconds are the rule, and that a drift rate ρ of the order of 10^{-5} or less makes terms of the form $d\rho$ -where d is a round trip delay- insignificant, allows us to simplify the formulae (6)-(9) as follows. When a slave S receives a successful round trip of length 2D from the master M, the master clock C_M is in the interval [T+min, T+2D-min]:

(6') $C_M(t) \in [T+min, T+2D-min]$.

By estimating the value of the master clock as being the midpoint of this interval

(7') $C_M^S(T,D) \equiv T+D$

the maximum reading error that S can make is

(8') $e = D-min$.

The protocol followed by a slave S relies upon the above simplified formulae. The remainder of this section presents this protocol informally and analyzes its behavior. A detailed description is given in the Appendix.

To keep synchronized with a master, a slave S attempts periodically to reach rapport. Each attempt at rapport consists of at most k attempts at reading the master clock, where successive reading attempts are separated by W clock time units. We assume W>2U, i.e. a slave knows whether its previous reading attempt has succeeded when it is time to try again reading. If during an attempt to reach rapport all k reading attempts fail, S must leave the group of synchronized slaves (such a departure can be followed by a later rejoin). Consider now that one of the reading attempts results in a round trip delay 2D<2U allowing S to reach rapport with M. At rapport, the speed of the slave logical clock C is set according to the equations (A) for the next t_a real time units, $(1-\rho)\alpha \leq t_a \leq (1+\rho)\alpha$, so that during amortization, say t real time units after rapport, $0 \leq t \leq t_a$, the worst case distance d between the slave clock C and the master clock is

(9') $d = (1-t/t_a)ms + t/t_a e + \rho t$,

where e is the reading error and *ms* is the worst case distance between C and C_M at rapport. The term ρt in (9') reflects the fact after rapport the slave clock C continues to drift from C_M. During amortization d is required to stay smaller than *ms*, i.e.

(10') $(1-t/t_a)ms + te/t_a + \rho t \leq ms$

By rewriting (10') we get

(11') $e+\rho t_a \leq ms$.

We show later that if amortization ends before a next attempt at rapport, (11') is satisfied.

Since the slave clock continues to drift from the master clock after amortization ends, it follows that for any $t \geq t_a$, the distance between C and C_M can be as large as $e+\rho t$. To keep C and C_M within *ms* of each other, i.e.

(12') $e+\rho t \leq ms$,

it is sufficient to ensure that after each rapport (with error e) the real time delay to the next rapport dnr is smaller than

(13') $dnr = \rho^{-1}(ms-e)$.

If at most k reading attempts are allowed (during which the slave S can drift from the master by as much as ρkw, where $w=(1+\rho)W$ is the maximum real time which can elapse between successive reading attempts), it follows that the maximum real time delay dna between a rapport and the next *attempt* at rapport must be

(14') $dna = \rho^{-1}(ms-e)-(1+\rho)kW$.

Since S must measure this delay with its own timer (which can run as fast as $1+\rho$), S must set the timer measuring the delay to the next attempt at rapport conservatively to

(15') $DNA = (1-\rho)dna = \rho^{-1}(1-\rho)(ms-e)-kW$.

Note that the time interval which elapses between a rapport and the beginning of the next attempt at rapport is *variable*, since it is a function of the round trip delay 2D observed at the last rapport. If D is close to min, the tight synchronization precision achieved allows the delay to the next attempt at rapport to be as long as:

(16') $DNA_{max} = \rho^{-1}(1-\rho)ms-kW$.

When rapport is achieved with a round trip delay that is barely acceptable (i.e. the reading error is close to U-min) the delay to the next attempt can be as short as

(17') $DNA_{min} = \rho^{-1}(1-\rho)(ms+min-U)-kW$.

We constrain amortization to end before a next attempt at rapport, i.e.

(18') $\alpha \leq DNA_{min}$.

Condition (18') implies (11'), that is, if amortization ends before a next attempt at rapport then C and C_M stay within *ms* during amortization. To keep logical clocks monotonic, the amortization period must also be chosen so that the speed change parameter m of (A) satisfies the relation m>-1. For this, it is sufficient to chose α greater than *ms*+U-min (see (23') for more details). Since the amortization parameter α is positive

(19') $0 \leq \alpha$.

we infer from (17') and (19') that

(20') $ms \geq U-min+\rho k(1+\rho)W$.

Thus, for a given choice of the U, k, and W constants, the smallest master slave maximum deviation that can be achieved is

(21') $ms_{min} = U-min+\rho k(1+\rho)W$.

For aggressive risky algorithms for which U is close to min, maximum deviations as small as $\rho k(1+\rho)W$ can be achieved at the expense of many synchronization messages (recall ρ is of the order of 10^{-5}). For sure "deterministic" algorithms for which U is close to an assumed maximum delay max, we get synchronization precisions slightly worse than max-min with only two messages per synchronization, a result comparable to the precisions achievable

by previously published deterministic synchronization algorithms [CAS], [DHSS], [L], [LL], [LM], [S], [ST].

The clock reading method described naturally tolerates communication failures: up to k-1 successive performance failures can be masked if they are followed by a successful rapport. The existence of variable delays between successive slave synchronizations is a useful property, since it will tend to uniformly spread the synchronization traffic generated by independent slaves in time.

PERFORMANCE: TWO NUMERICAL EXAMPLES

To illustrate the synchronization precisions achievable by our time service, we analyze in this section its performance in the context of a simple system of two 4381 processors [D], assuming one plays the role of master and the other one the role of slave.

If we chose 2U to be the median round trip delay 2U=4.48 milliseconds, the probability p of an unsuccessful round trip is 0.5. By (12) this yields an average number $\bar{n}$ of messages per successful rapport of $\bar{n}$=4. Assuming that a probability of losing synchronization of 10^{-9} is acceptable, we find that at least k=30 successive attempts at rapport should be allowed ($(0.5)^{30} < 10^{-9}$). Assuming a worst case drift rate of $\rho=6*10^{-6}$, a waiting time constant between successive reading attempts W of 2 seconds, formulae (16') and (17') indicate that it is possible to achieve a maximum master slave deviation *ms* of 1 millisecond. The minimum, average, and maximum delays between successive synchronizations are 63, 67, and 108 seconds, respectively. Thus, for this choice of U, ρ, k, and W, a slave stays within a maximum deviation of *ms*=1 millisecond from a master with probability greater than $1-10^{-9}$ by sending on the average $\bar{n}$ =4 messages every 1.11 minutes.

A more conservative choice of 2U'=5.2 milliseconds, yields a probability p' of an unsuccessful round trip of 0.05 and an average number $\bar{n}$ ' of messages per successful rapport of 2.1. For this p', to achieve a probability of successful rapport greater than $1-10^{-9}$, k must be chosen to be at least 7 (i.e. $((0.05)^7 < 10^{-9})$. Since for this choice of U', the reading error is 0.98 milliseconds, we settle for the goal of achieving a maximum master slave deviation of *ms*'= 2 milliseconds. Assuming as in the previous example $\rho=6*10^{-6}$, and W= 2 seconds, we find that to achieve the 2 milliseconds maximum deviation, a slave must on the average spend 2.1 messages to reach rapport with a master every 231 seconds (3.85 minutes). The minimum and maximum delays between successive resynchronizations is 230 and 273 seconds, respectively.

The above example precisions compare favorably with the best precision of at most 44.47 milliseconds achievable by the deterministic synchronization algorithms described in [CAS], [DHSS], [LM], [LL], [M], [S], [ST].

EXTENSIONS

In this section we relax two of the assumptions made earlier: the existence of a continuously available master processor and the existence of reliable clocks that never fail. We mention how to handle slave failures and adapt to variable system load. In [Cr] we also deal with the important problems of improving the accuracy with which synchronized clocks measure the passage real-time and reducing synchronization related message traffic by taking advantage of past statistics on round trip message delays. The main idea is to use these statistics to estimate the actual drift ρ_s of hardware clocks, and use this estimates to build, at a higher level of abstraction, *self-adjusting* virtual hardware clocks that run at speeds very close to 1 by automatically compensating for the drift of the underlying real hardware clocks.

Dealing with master server failures

With a unique master server, the master time service fails when the unique process implementing it fails. The probability of a master service failure can be reduced if the service is implemented by a group of *redundant* master servers, all synchronized within *em* of external time. There are several ways in which such a group can be structured. We mention three alternatives.

Active Master Set. In this arrangement each slave multicasts ("time=?") requests to all masters, each master answers each time request, and slaves pick up the first answer that arrives. With such a strategy, synchronized slaves will stay within *es* =*em*+*ms* of external time, but since some slaves might be synchronized to one master, and some others to another, the relative deviation among slaves *ss* becomes 2*es*, instead of 2*ms* as before. This solution leads to an increase in message cost: 2m messages per attempt at rapport, where m is the number of members in the master group. Note, however, that if all processors are on a broadcast local area network, this number can be reduced to m+1.

Ranked Master Group. To reduce the message cost, one can use a synchronous membership protocol [C] to rank the group of active masters into a primary synchronizer, back-up, and so on. With such an arrangement slaves would send their requests only to the primary. This results in a message cost per attempt at rapport of 2. Let C be the upper bound on the failure detection time guaranteed by the synchronous master membership protocol (C is a function of the partition detection timeout delay maxp that is chosen [C]) and let A be the time needed to inform the slaves that all subsequent time requests should be sent to a new master. If W is chosen greater than C+A, a slave cannot distinguish between a master failure and an excessive synchronization message delay, so the maximum deviations *es* and *ss* stay as before, i.e. *es*=*em*+*ms*, *ss*=2*es*. If W is chosen smaller than C+A, then one has to adopt a higher upper bound on the maximum number of successive attempts at rapport and the analysis of the probability of achieving rapport becomes slightly more complex.

Active Master Ring. A third solution would use a master membership protocol to order all active masters on a virtual ring. To send a time request a slave chooses an active master at random. If no answer arrives within 2U, the slave asks the next active master on the ring, and so on. The message cost of each attempt at rapport is 2 as in the *Ranked Master Group* case, but the maximum deviations *es* and *ss* stay the same as in the *Active Master Set* architecture, irrespective of the relation between W and C+A, where A corresponds in this case to the worst case time needed to inform all slaves of a master group membership change.

Detecting clock failures

Let U_M, min_M be the parameters of the probabilistic algorithm run by a master M to synchronize its clock C_M to the clock C_R of its

attached receiver R. The maximum difference which can exist between C_M and M's estimate of C_R at rapport is

(22') $\quad maxadj_M = e_M + em$.

The first term $e_M = U_M - min_M$ represents the maximum error in reading C_R while the second term accounts for the maximum distance which might exist between C_R and C_M at rapport. If a previously synchronized master M detects at rapport that the distance between C_M and its estimate of C_R is greater than $maxadj_M$, a master clock failure has occurred (recall our assumption that the source of external time signals is reliable and that receivers are self-checking). Upon detecting the failure of its local clock, a master server leaves the active master group after reporting the failure to an operator.

Similarly, if a master M and a slave S are correct and synchronized within *ms*, the maximum difference at rapport between the clock C of the slave and the slave's estimate of C_M is

(23') $\quad maxadj_S = (U - min) + ms + 2em$.

The last term is added because S can successively synchronize to different masters that are *2em* apart from each other. A slave which at rapport detects that its clock is more than $maxadj_S$ apart from a master, detects either a master or a local clock failure. Assuming that masters synchronize more frequently than slaves, if a master clock failure has occurred, the master will detect the failure before the next rapport with the slave. Thus, a slave detects a local clock failure if it observes twice in a row that its distance to the same master clock is greater than $maxadj_S$. Upon detecting the failure of its clock, a slave leaves the group of synchronized time servers after reporting the failure to an operator.

Dealing with slave server failures

A slow slave, which takes too long to read its messages or to time out, eventually discovers that the distance between its clock and a master clock has become unacceptable when it evaluates the test (23'). Early slave timing failures (e.g. caused by fast slave timers) lead to an increase in network synchronization traffic. The occurrence of such failures can be detected by the master group, if the masters keep track of the last time each slave has asked for the time. If a slave asks too often, the masters could simply ignore it. The faulty slave will then fail to synchronize and will eventually leave the group of synchronized servers.

Adapting to changing system load

Another extension consists in making the choice of the actual U-indexed slave synchronization algorithm used in a system at a given moment dependent on the system load. We sketch here a possible way to take into consideration system load when deciding on a round trip acceptability threshold U. The intention is that U should increase when load increases and should decrease when load decreases. This can be achieved in the following way. If *at least one* slave experiences k'<k successive unsuccessful attempts at reaching rapport with a master, it should announce to the master group that all slaves have to adopt a higher timeout delay U' (U'>U) known in advance. The masters could then agree on this and diffuse a decision that beginning with some time in the future everybody has to switch to the new round trip acceptability threshold U' and, hence, to bigger master-slave maximum deviations. The effect will be to increase the maximum master slave clock deviation when the system load increases.

To decrease the maximum clock deviation when load is light, a slave processor could communicate to the master group the fact that it measures round trip delays that are consistently smaller than U", for some U"<U known in advance. If the masters receive such messages from *all* slave processors, they could diffuse the information that beginning with some time in the future, everybody should decrease their round trip acceptability threshold to U". The effect will be to decrease the master slave clock deviation when the system load decreases.

CONCLUSION

A new probabilistic approach for reading remote clocks was proposed and illustrated by presenting a synchronization algorithm that achieves external clock synchronization. The new approach allows one to achieve precisions better than the best precision bound (max-min)(1-1/n) guaranteeable by previously published deterministic algorithms [CAS], [DHSS], [L], [LL], [LM], [LWL], [S], [ST]. (Specialized hardware can be used to reduce the difference between max and min [KO], but the inherent limitation of deterministic protocols remains unchanged.) When indexed by a conservative parameter U, such as a partition detection timeout delay maxp, our external clock synchronization algorithm also achieves a relative deviation at rapport 2(maxp-min) smaller than the best precision 4(maxp-min) achievable by previously published algorithms based on partition detection timeouts [GZ], [M]. One of the key observations of this paper is that no relation needs to exist between clock synchronization and partition detection timeout delays. Synchronization algorithms indexed by timeout parameters U close to min can in theory achieve synchronization precisions close to $3\rho min$, where ρ and min are of order of 10^{-5} and 10^{-3} seconds, respectively, for commercially available clocks and local area networks. One can envisage that by estimating actual clock drift rates and using self-adjusting clocks as suggested in [Cr], one could achieve precisions even better than the above bound.

Besides improving synchronization precision, the new approach has other properties worth mentioning. Since a probabilistic approach does not assume an upper bound on message transmission delays, it can be used to synchronize clocks in all systems, not only those which guarantee an upper bound on message delays. A probabilistic time service such as the one sketched previously distributes uniformly the clock synchronization traffic in time, avoiding the periodic synchronization traffic bursts produced by the existence of synchronization points in previously known synchronization algorithms. The service is simple to implement (see Appendix) and robust. Likely process and communication failures are tolerated. Clock failures are detected and processes with faulty clocks are shut down. Finally the time service described is efficient: it uses a number of messages that is linear in the number of processes to be synchronized.

While deterministic synchronization protocols *always* succeed in synchronizing clocks, the probabilistic approach proposed in this paper carries with it a certain *risk* of not achieving synchronization. In view of the impossibility result of [LL] that deterministic clock synchronization algorithms cannot synchronize the clocks of

n processes closer than (max-min)(1-1/n), this seems to be an unavoidable price for wanting to achieve a higher precision. As the desired precision becomes higher, more messages are sent and the risk of not achieving synchronization becomes higher. Conversely, as the desired precision becomes lower, the risk of not achieving synchronization becomes lower and fewer messages need to be sent. Actually, the U-indexed family of master-slave synchronization algorithms presented achieves a *continuum* between, at one end, sure "deterministic" protocols (indexed by large Us close to maxp or max) that achieve poor precision with a high probability and a small number of messages, and "aggressive" protocols (indexed by small Us close to min) capable of achieving very high precision but which carry with them a significant risk of not achieving synchronization even when substantial numbers of messages are exchanged. In practice, one needs to choose a parameter U that achieves the right balance between precision and message overhead, and reduces the risk of losing synchronization to a level that is acceptably small.

The new view cast on clock synchronization in this paper prompts a number of questions which can lead to further research. We mention here several. How is it possible to improve the accuracy of some of the internal clock synchronization algorithms surveyed in [S] by using probabilistic remote clock reading methods? What lower bounds exist for probabilistic synchronization algorithms? How can the accuracy of clock synchronization be improved if one knows the distribution obeyed by message delays? How can one estimate bounds on the actual drift rate of hardware clocks that are better than the manufacturer specified bound ρ? (An answer to this question for the case when the actual clock drift is a constant is given in [Cr], but other cases also need to be considered, e.g. the actual drift rate of clocks is a time varying function possessing a first derivative bounded by constants from above and below). And finally, how can one design algorithms which adapt to variable system load?

ACKNOWLEDGEMENTS

The idea that the randomness inherent in message transmission delays can be used to improve clock synchronization precision originated while the author was working with John Rankin and Mike Swanson on problems related to synchronizing the clocks of high end IBM processors in Poughkeepsie, NY, during February 1986. In the summer of 1987, John Palmer independently proposed a similar clock synchronization protocol for the high end Amoeba system prototype under development at the Almaden Research Center. The round trip delay measurements used in this paper were performed by Margaret Dong in the Amoeba system. Discussions with Danny Dolev, Frank Schmuck, Larry Stockmeyer, and Ray Strong helped improve the contents and the form of this exposition. The research presented was partially sponsored by IBM's Systems Integration Division group located in Rockville, Maryland, as part of a FAA sponsored project to build a new air traffic control system for the US.

REFERENCES

[C] F. Cristian: Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems", 18th Int Conf on Fault-Tolerant Computing, Tokyo, Japan, June 1988.

[CAS] F. Cristian, H. Aghili, R. Strong: Approximate Clock Synchronization despite Omission and Performance Faults and Processor Joins, 16th International Symposium on Fault-Tolerant Computing, Vienna, Austria, 1986.

[CASD] F. Cristian, H. Aghili, R. Strong, D. Dolev: Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement, 15th International Symposium on Fault-Tolerant Computing, Ann Arbor, Michigan, June 1985.

[Cr] F. Cristian: Probablistic Clock Synchronization, IBM Research report RJ6432, Revised version, March 1989.

[D] M. Dong, private communication, June 1988.

[DHSS] D. Dolev, J. Halpern, B. Simons, R. Strong: Fault-Tolerant Clock Synchronization, Proc of the 3d ACM Symposium on Principles of Distributed Computing, 1984.

[GZ] R. Gusella, S. Zatti: The Accuracy of the Clock Synchronization Achieved by Tempo in Berkeley Unix 4.3BSD. Report UCB/CSD 87/337, 1987.

[K] Kinemetrics/Truetime: Time and Frequency Receivers, Santa Rosa, California, 1986.

[KO] H. Kopetz, W. Ochsenreiter: Clock Synchronization in Distributed Real-Time Systems, IEEE Tr. on Comp., Vol. C-36, NO. 8, 1987.

[L] L. Lamport: Synchronizing Time Servers, TR 18, DEC Systems Research Center, Palo Alto, California, June 1987.

[LL] J. Lundelius, N. Lynch: An upper and Lower Bound for Clock Synchronization, Information and Control, Vol. 62, No. 2-3, 1984, pp. 190-204.

[LM] L. Lamport, M. Melliar-Smith: Synchronizing Clocks in the Presence of Faults, Journal of the Association of Computing Machinery, Vol. 32, No. 1, January 1985, pp. 52-78.

[LWL] J. Lundelius-Welch, N. Lynch: A New Fault-tolerant Algorithm for Clock Synchronization, Information and Computation, Vol. 77, No. 1, 1988, pp. 1-36.

[M] K. Marzullo: Maintaining the Time in a Distributed System, Xerox report OSD-T8401, March 1984.

[S] F. Schneider: Understanding Protocols for Byzantine Clock Synchronization, TR 87-859, Cornell Univ., August 1987.

[ST] T.K. Srikanth, S. Toueg: Optimal Clock Synchronization, JACM, Vol. 34, No. 3, July 1987, pp. 626-645.

APPENDIX: DETAILED DESCRIPTION OF THE MASTER SLAVE PROTOCOL

A detailed description of a slave time server under the simplifying unique master assumption is given in Figures 2 and 3. To simplify this presentation, we do not give a detailed description of the master time server, since it is similar to a slave server and we do not use self-adjusting logical slave clocks. In what follows, we refer to line j of figure i as (i.j).

The round trip acceptability threshold U, the maximum number of successive reading attempts k, waiting time between reading attempts W, and the amortization delay α, and the maximum deviation ms are parameters of the protocol (2.1). Once U is chosen,

```
1  task Slave(U:Time, k:Int, W,α,ms:Time);
2  const min: Time;
3     master: processor;
4  var T,T',D,N: Time; try,sn: Integer; m: Real;
5     Synch,Attempt,Amort: Timer;
6     synchronized: Boolean; H: hardware-clock;

7  sn←0; synchronized←false; Synch.set(0);
8  cycle
9    when lreceive("time?") do send-local-time;
10   when Synch.timeout
11   do try←1; sn←sn+1; T'←H;
12      send("time=?",sn) to master; Attempt.set(W);
13   when receive("time=",ET,sn')
14   do if sn≠sn' then iterate fi;
15      T←H; D←(T-T')/2;
16      if D>U then iterate fi;
17      Attempt.reset; compute-adjustment;
18      Synch.set(ρ⁻¹(1 − ρ)(ms+min-D)-kW));
19   when Attempt.timeout
20   do if try≥k then synchronized←false; "leave" fi;
21      try←try+1; sn←sn+1; T'←H;
22      send("time=?",sn) to master; Attempt.set(W);
23   when Amort.timeout do m←0; N←N-HC;
24 endcycle
```

Figure 2.

the probability p that, under worst case load conditions, a round trip delay is greater than 2U is also determined. The constant k must be chosen to ensure that the probability p^k of observing k successive round trip delays greater than 2U is acceptably small (typically two or more orders of magnitude smaller then the instantaneous crash rate of the underlying processor). We assume that the constant W is chosen greater than the acceptable round trip delay 2U.

```
1  procedure send-local-time;
2  if synchronized
3  then lsend("time=",H+N+m*H)
4  else lsend("undefined")
5  fi;

6  procedure compute-adjustement;
7  if synchronized
8  then if ¬okadj then synchronized←false; "leave" fi;
9       m←((ET+D)-(T+N+m*T))/α;
10      N←N-m*T; Amort.set(α);
11 else m←0; N←(ET+D)-T;
12      synchronized←true;
13 fi;
```

Figure 3.

The slave protocol uses three timers (2.5): a "Synch" timer for measuring delays between successive synchronization attempts, an "Attempt" timer for measuring delays between successive master clock reading attempts, and an "Amort" timer for measuring amortization periods. There are two kinds of operations that are defined on a timer T: set and reset. The meaning of a T.set(δ) invocation is "ignore all previous T.set invocations and signal a T.timeout event δ clock time units from now". The meaning of a T.reset invocation is "ignore all previous T.set invocations". Thus, if after invoking T.set(100) at local time 200, a new T.set(100) invocation is made at local time 250, there is no timeout event at time 300. If no other T.set or T.reset invocation is made before time 350, a timeout event occurs at local time 350.

A local Boolean variable "synchronized" (2.6) is true when the local clock is synchronized to the master clock and is false when the local clock can be out of synchrony with respect to the master clock. The local logical time is undefined when "synchronized" is false (3.4). To prevent any confusion between messages pertaining to old and current clock reading attempts in the presence of performance failures each attempt is uniquely identified by a sequence number "sn" (2.4). Another variable "try" counts the number of unsuccessful master clock reading attempts (2.4).

```
1  task Master;
4  var N: Time; m: Real; H: Hardware-clock;

2  cycle
3  when receive("time=?",sn) from s
4  do send("time=",H+N+m*H,sn) to s
5  endcycle;
```

Figure 4.

After initializing the "sn" and "synchronized" local variables, an attempt to synchronize with the master is immediately scheduled (2.7). When the Synch.timeout event occurs (2.10), the counter for unsuccessful attempts "try" is initialized and a message numbered "sn" is sent to the master (2.12). The master responds to this message by sending its clock value (4.3). To simplify our presentation, we assume that a master clock is always synchronized to external time (the absence of synchrony with external time at the master can be handled in a manner similar to the absence of synchrony with the master at a slave). When a master response arrives (2.13), if the received sequence number matches the local sequence number (2.14) the message is accepted, otherwise it is discarded (the *iterate* command terminates the current iteration of the loop (2.8-2.24) and begins a new iteration). Unacceptably long round trips are discarded (2.16). Unsuccessful reading attempts cause Attempt.timeout events (2.19). Indeed, the "Attempt" timer is set by each new attempt at reaching rapport (2.12, 2.22) and is reset only when rapport is reached (2.17). If k successive unsuccessful attempts occur (2.20), a slave can no longer be sure that its clock is within *ms* from the master clock and must leave the group of synchronized slaves. Such a departure can be followed by a later rejoin.

Consider now that a matching answer which arrives in less than 2U time units leads to a successful rapport and causes the "Attempt" timer to be reset (2.17). If the slave logical clock was not previously synchronized, it is bumped to the estimate (7') of the master time (3.11). If the slave was previously synchronized, and the adjustment to be made passes the reasonableness test "okadj" (3.8) defined in (23'), the speed of the local logical clock C is set so as to reach the slave's estimate of the master clock within α clock time units (3.9-3.10) following equation (A). After amortization ends (2.23) the logical slave clock is let again to run at the speed of the hardware clock until the next rapport (2.17).