The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD

RICCARDO GUSELLA, STUDENT MEMBER, IEEE, AND STEFANO ZATTI, MEMBER, IEEE

Abstract—**We discuss the upper and lower bounds on the accuracy of the time synchronization achieved by the algorithm implemented in TEMPO, the distributed service that synchronizes the clocks of Berkeley UNIX® 4.3BSD systems. We show that the accuracy is a function of the network transmission latency, and depends linearly upon the drift rate of the clocks and the interval between synchronizations. Comparison with other clock synchronization algorithms reveals that TEMPO may achieve better synchronization accuracy at a lower cost.**

Index Terms—**Clock synchronization, distributed systems, fault-tolerance, master-slave, time service.**

I. INTRODUCTION

THIS paper discusses the upper and lower bounds on the accuracy of the time synchronization achieved by the algorithms implemented in TEMPO, a distributed clock synchronizer running on Berkeley UNIX 4.3BSD systems.

TEMPO, which works in a local area network, consists of a collection of *time daemons* (one per machine) and is based on a master-slave structure [3], [4].

Figs. 1–4 sketch the way TEMPO works. A *master time daemon* measures the time difference between the clock of the machine on which it is running and those of all other machines. The master computes the *network time* as the average of the times provided by nonfaulty clocks. A clock is considered faulty if its value is more than a small specified interval away from the values of the clocks of the majority of the other machines. (The clock of Slave 3 in Fig. 2 is faulty.) The master then sends to each *slave time daemon*, also to those with faulty clocks, the correction that should be performed on the clock of its machine. Since the correction can be negative, in order to preserve the monotonicity of the clocks' time functions, TEMPO implements it by slowing down (or speeding up) the clock rates [1]. This process is repeated periodically. Because the correction is expressed as a time difference rather than an absolute time, transmission delays do not interfere with synchronization.

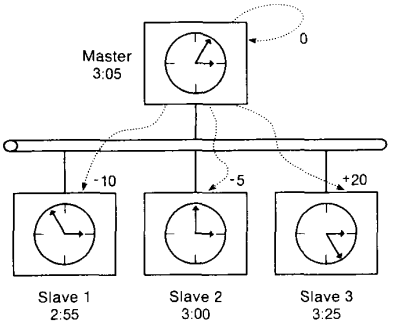
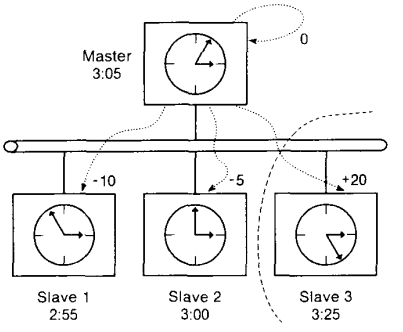


Fig. 1. The measurements.



$$Av = \frac{0 - 10 - 5}{3} = -5$$

Fig. 2. The computation of the average.

When a machine comes up and joins the network, it starts a slave time daemon, which asks the master for the correct time and resets the machine's clock before any user activity can begin. TEMPO therefore maintains a single network time in spite of the drift of clocks away from each other.

An election algorithm that elects a new master should the machine running the current master crash, the master terminate (for example, because of a run-time error), or the network be partitioned, ensures that TEMPO provides continuous, and therefore reliable service [5]. However, in the following discussion we will assume that elections do not occur, as we are only concerned with determining the accuracy achieved by the clock synchronization algorithms.

Manuscript received March 21, 1987. This work was supported by the Defense Advanced Research Projects Agency (DoD), Arpa Order 4871 monitored by the Naval Electronics Systems Command under Contract N00039-84-C-0089, and by the CSELT Corporation.

R. Gusella is with the Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

S. Zatti is with IBM Research, Zurich Laboratory, Säumerstrasse 4, CH-8803 Rueschlikon, Switzerland.

IEEE Log Number 8928285.

®UNIX is a registered trademark of AT&T Bell Laboratories.

0098-5589/89/0700-0847$01.00 © 1989 IEEE

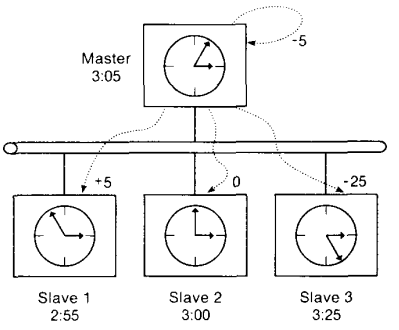


Fig. 3. The correction of the clocks.

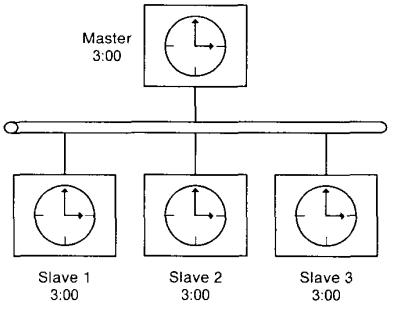


Fig. 4. Clocks are now synchronized.

II. Definitions and General Assumptions

A physical clock generates an approximation, as precise as possible, of t, the universal Galilean time. A real-valued, continuous, and everywhere derivable function $C(t)$ describes its behavior. Let ρ be the absolute value of the maximum drift rate over all clocks from the universal time; we have

$$1 - \rho \leq \frac{dC(t)}{dt} \leq 1 + \rho. \tag{1}$$

Two clocks are said to be *synchronized* at time t_0 if their associated functions have the same value, i.e., in case of clocks of machines A and B, if $C_A(t_0) = C_B(t_0)$.

Let R be a constant. Two or more clocks are *within range R at time* t_0 if the difference between any two of them is bounded by R:

$$\left|C_A(t_0) - C_B(t_0)\right| \leq R.$$

Lemma 1: For $t_1 \geq t_0$

$$(1-\rho)(t_1 - t_0) \leq C(t_1) - C(t_0) \leq (1+\rho)(t_1 - t_0).$$

Proof: Immediate by integrating (1). □

Lemma 2: The absolute value of the relative drift rate of any two clocks satisfying (1) is at most 2ρ

$$\left|\frac{d\left(C_A(t) - C_B(t)\right)}{dt}\right| \leq 2\rho.$$

Proof: From (1), we have for C_A

$$1 - \rho \leq \frac{dC_A(t)}{dt} \leq 1 + \rho$$

and for C_B

$$-1 - \rho \leq -\frac{dC_B(t)}{dt} \leq -1 + \rho.$$

Adding term by term we obtain

$$-2\rho \leq \frac{dC_A(t)}{dt} - \frac{dC_B(t)}{dt} \leq 2\rho.$$

Lemma 2 follows. □

A direct consequence of Lemma 2 is that, if two clocks are synchronized at time t_0, at any later time t_1 their values can differ at most by $\pm 2\rho(t_1 - t_0)$.

III. The Clock Difference Measurement Algorithm

A time daemon program on machine A measures the time difference between the clock of machines A and B by timestamping a message at time $C_A(t_1)$ and sending it to machine B. The kernel of machine B timestamps that messages at time $C_B(t_2)$ and sends it back.[1] Upon receipt of the message from machine B, the time daemon reads the time $C_A(t_3)$. This process is represented in Fig. 5. As derived in Theorem 1 below, the time daemon can then estimate $\Delta_{AB}(t)$, the difference between the clocks of machines A and B, as

$$\frac{C_A(t_1) + C_A(t_3)}{2} - C_B(t_2).$$

As indicated, Δ_{AB} is a function of time, but we assume that its variation in the interval $t_3 - t_1$ is so small that we can write

$$\Delta_{AB}(t_3) = \Delta_{AB}(t_1) = \Delta_{AB}.$$

Also, notice that $\Delta_{BA} = -\Delta_{AB}$.

Theorem 1: Let $T_{m_{AB}}$ and $T_{m_{BA}}$ be the minimal possible transmission times from A to B and from B to A, respectively. Let us fix a bound, $T_M \geq 2 \max(T_{m_{AB}}, T_{m_{BA}})$, on the round-trip time, i.e., $C_A(t_3) - C_A(t_1) \leq T_M$. Then, the maximum error in the estimation of Δ_{AB} is

$$\epsilon = \frac{T_M - 2\min(T_{m_{AB}}, T_{m_{BA}})}{2} \geq 0. \tag{2}$$

Proof: Let $T_{S_{AB}}$ and $T_{S_{BA}}$ be the actual transmission times from A to B and vice versa. We have

$$T_{m_{AB}} + T_{m_{BA}} \leq T_{S_{AB}} + T_{S_{BA}} \leq T_M,$$

[1]TEMPO implements this exchange of messages using the *TimeStamp* and *TimeStampReply* messages of the DARPA Internet Control Message Protocol (ICMP) [11]. As soon as the associated interrupt of the network interface is served, the kernel of a remote machine processes a TimeStamp message by changing its type field to TimeStampReply, writing the clock value in the message, and sending it back without invoking a user process. It is simply a variant of an *echo* protocol. We can therefore consider that the remote time query occurs *instantaneously* at the remote machine at time t_2.

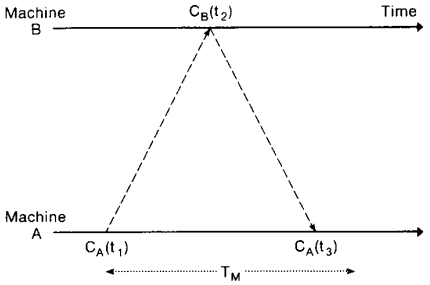


Fig. 5.

and also

$$\max(T_{S_{AB}}) = T_M - T_{m_{BA}},$$
$$\max(T_{S_{BA}}) = T_M - T_{m_{AB}} \tag{3}$$

for the hypotheses.

We can now compute

$$\delta_1 = C_B(t_2) - C_A(t_1) = -\Delta_{AB} + T_{S_{AB}},$$
$$\delta_2 = C_A(t_3) - C_B(t_2) = \Delta_{AB} + T_{S_{BA}},$$

and, if E_{AB} is our estimate of Δ_{AB},

$$E_{AB} = \frac{\delta_2 - \delta_1}{2} = \frac{C_A(t_1) + C_A(t_3)}{2} - C_B(t_2)$$
$$= \Delta_{AB} + \frac{T_{S_{BA}} - T_{S_{AB}}}{2}. \tag{4}$$

From (3) we can derive

$$-(T_M - 2T_{m_{BA}}) \le T_{S_{BA}} - T_{S_{AB}} \le (T_M - 2T_{m_{AB}}). \tag{5}$$

By substituting (5) into (4), we get

$$\Delta_{AB} - \frac{T_M - 2T_{m_{BA}}}{2} \le E_{AB} \le \Delta_{AB} + \frac{T_M - 2T_{m_{AB}}}{2}. \tag{6}$$

If we define

$$\epsilon = \frac{T_M - 2\min(T_{m_{AB}}, T_{m_{BA}})}{2} \ge 0,$$

the relations

$$\epsilon \ge \frac{T_m - 2T_{m_{AB}}}{2} \text{ and } \epsilon \ge \frac{T_M - 2T_{m_{BA}}}{2}$$

hold for the definition of T_M, and the theorem, which can also be written as

$$|E_{AB} - \Delta_{AB}| \le \epsilon, \tag{7}$$

follows. □

In general $T_{m_{AB}}$ and $T_{m_{BA}}$ will be different, as in the case of a ring network where the information flow moves in the same direction. However, these two times can also be different in a bus network because, for example, of different interrupt structures of the two machines.

In the actual implementation, several round-trip messages are exchanged and the minimum values of δ_1 and δ_2 are used in the computation of E_{AB}. This reduces the variances of the transmission times in the two directions and provides a better estimate of Δ_{AB}.

If the estimate E_{AB} is used to correct the clock of machine B, the two machines' clocks are, upon synchronization, within range ϵ.

Corollary 1: The lower bound for the error ϵ is $|T_{m_{AB}} - T_{m_{BA}}|$.

Proof: Immediate by substituting the expression for T_M into (2). □

Corollary 2: The measurement algorithm allows a time daemon to compute the clock difference between any two other machines with maximum error 2ϵ.

Proof: Let us suppose that A sends clock difference measurement messages to any two machines, for instance machines B and C, then

$$\Delta_{AB} = C_A(t) - C_B(t), \quad E_{AB} = \Delta_{AB} \pm \epsilon,$$
$$\Delta_{AC} = C_A(t) - C_C(t), \quad E_{AC} = \Delta_{AC} \pm \epsilon,$$
$$\Delta_{BC} = \Delta_{AC} - \Delta_{AB}, \quad E_{BC} = E_{AC} - E_{AB}.$$

It follows

$$E_{BC} = \Delta_{AC} - \Delta_{AB} \pm 2\epsilon = \Delta_{BC} \pm 2\epsilon. \quad \square$$

IV. The Synchronization Algorithm

The master time daemon, using the clock difference measurement algorithm, computes the time differences between its clock and the clocks of slave machines. In order to prevent malfunctioning clocks as well as clocks with abnormally large drift rates to adversely affect other clocks, a fault-tolerant averaging function is applied to these differences. It selects the largest sets of clocks that do not differ from each other more than a small quantity γ and averages the differences of these clocks. For instance, in the example of Figs. 1, 2, 3, and 4, assuming that γ is 10 minutes, the fault-tolerant function selects the set consisting of the clock of the Master, the clock of Slave 1, and that of Slave 2. Clocks that are not selected by the fault-tolerant function are considered faulty. Last, the master time daemon asks each slave to correct its clock by a quantity equal to the difference between the average value provided by the fault-tolerant function and the previously measured difference between the clocks of the master and the slave machines. (The master time daemon, as suggested in Fig. 3, also corrects the clock of the machine on which it runs using the same method.) This process is repeated every T seconds. Notice that the synchronization algorithm produces the appropriate correction value for every clock, including the faulty ones.

For TEMPO to be reliable, it is necessary that all properly functioning clocks be within γ seconds when the master starts a synchronization round. The constant γ is therefore chosen as a function of the clock drift rate; the interval between synchronization rounds, T; and the measurement errors as derived in Theorem 3 below.

Theorem 2: If the master, using the synchronization algorithm described above synchronizes a number of ma-

chines, then any two nonfaulty clocks are, once the synchronization is performed, within range 4ϵ.

Proof: Let Q be the set of machines selected by the fault-tolerant averaging function and $|Q|$ its cardinality. The average of the measurements is then

$$\frac{1}{|Q|}\sum_{J\in Q} E_{AJ} = \frac{1}{|Q|}\sum_{J\in Q} \Delta_{AJ} \pm \left(\frac{|Q|-1}{|Q|}\right)\epsilon, \quad (8)$$

where we have assumed that the clock of the master A is also nonfaulty[2] and $\Delta_{AA} = 0$ by definition. (This implies that also $E_{AA} = 0$.)

If we use the symbols $\overline{\Delta}$ for $(1/|Q|)\,\Sigma_{J\in Q}\,\Delta_{AJ}$ and $\overline{E}$ for $(1/|Q|)\,\Sigma_{J\in Q}\,E_{AJ}$ in order to simplify the notation, we can rewrite (8) as

$$|\overline{E} - \overline{\Delta}| \le \epsilon' \quad (9)$$

where we have defined $\epsilon' = \epsilon$.

The correction performed on the clock of machine K is

$$c_K = \overline{E} - E_{AK},$$

from which, by adding the quantity $\Delta_{AK} - \overline{\Delta}$, and for (7) and (9) we obtain

$$|c_K + \Delta_{AK} - \overline{\Delta}| \le |\overline{E} - \overline{\Delta}| + |\Delta_{AK} - E_{AK}|$$
$$\le \epsilon' + \epsilon.$$

Let us represent with Δ'_{BC} the difference between the clocks of machines B and C after the correction is made

$$\Delta'_{BC} = (\Delta_{AC} + c_C) - (\Delta_{AB} + c_B).$$

By adding and subtracting $\overline{\Delta}$ we can write

$$\Delta'_{BC} = (\Delta_{AC} + c_C - \overline{\Delta}) - (\Delta_{AB} + c_B - \overline{\Delta}),$$

and also

$$|\Delta'_{BC}| \le |c_C + \Delta_{AC} - \overline{\Delta}| + |\overline{\Delta} - c_B - \Delta_{AB}|$$
$$\le 2\epsilon' + 2\epsilon = 4\epsilon,$$

which completes the proof. □

The following theorem summarizes the previous results.

Theorem 3: If the master time daemon synchronizes every T seconds a set of machines using the algorithm above described, then, at any time, all nonfaulty clocks are within range $4\epsilon + 2\rho T$.

Proof: The first term, 4ϵ, as per Theorem 2, accounts for the inaccuracy of synchronization after the clocks have been reset. The second term, as per Lemma 2, accounts for the maximum possible drift of any two clocks during the time betwen two subsequent synchronizations. □

[2]This is not a necessary assumption. The algorithm and the derivations are valid irrespective of whether or not the master's clock is selected by the fault-tolerant averaging function. Refer, however, to the next section of this paper for a brief discussion of the types of faults that TEMPO can tolerate.

V. Discussion

It is important to notice that in the derivation of the bounds on the time accuracy we have made no assumption whatsoever about the statistical distribution of the transmission times between two machines, nor have we assumed that these distributions are the same in the two directions of communication.

It should also be noted that the requirements on the maximum round-trip time T_M can be verified by the master, in the notation used above, by computing $C_A(t_3) - C_A(t_1)$. Even though messages can be arbitrarily delayed, the master is always able to reject measurements that do not satisfy the conditions of Theorem 1.

In our implementation of TEMPO for the Ethernet local area network, we have chosen a value of 20 milliseconds for T_M. Although the Digital Equipment VAX Hardware Handbook states that ρ can be as high as 10^{-4}, we have verified, using a high-resolution frequency meter, that the clocks of the VAX's used in our experiments display drift rates smaller than 2 parts in 10^5. Since the minimum transmission delay from machine to machine can be estimated to be 5 milliseconds (including kernel protocol handling and the scheduling delays of the master process), and since TEMPO synchronizes the clocks every 4 minutes, the maximum error in Theorem 3 is 30 milliseconds.

Let us call ϵ_{AB} the *actual* error in the measurement of the clock difference between machines A and B. From (6) we have $-\epsilon \le \epsilon_{AB} \le +\epsilon$. Therefore, the actual quantity that corresponds to ϵ' in (9) is, from (8) $(1/|Q|)\,\Sigma_{J\in Q}\,\epsilon_{AJ}$, that is the average of the actual errors of the measurements between the master A and the other machines in the set Q. As such, by the Strong Law of Large Numbers, this quantity converges in probability to the mean of the random variables that models the measurement errors. Under the condition of identically distributed transmission times in the two communication directions, which is satisfied in the case of the Ethernet, this mean, as can be recognized in (6), is zero. While according to Theorem 3 the first component of the global error can be as large as 4ϵ, the algebraic manipulations in the proof of Theorem 2 show that it can be separated into two parts, one of which, $2\epsilon'$, for what we have just seen, should be very small.

In measurements taken in our environment, where the time daemons synchronized the clocks of about 15 machines, we rarely found the time difference between clocks to be larger than 25 milliseconds, with the mean between 18 and 20 milliseconds. Since the drift rate of the clocks makes them diverge at most 10 milliseconds in 4 minutes, we estimated that the synchronization inaccuracies due to

the error described in Theorem 2 amount to about 10 milliseconds on the average.

As previously observed, a clock is considered faulty if it is not selected by the fault-tolerant averaging function. Therefore, great attention must be paid to the appropriate choice for the value of γ. If γ is too small, only a few clocks may be selected; if it is too large, malfunctioning clocks can reduce the precision of the synchronized time. In both cases, the reliability of TEMPO decreases. Since our measurements showed that most clocks do not diverge more than 20 milliseconds from each other, we set γ equal to 20 milliseconds.

The fault-tolerant averaging function may reject a clock measurement for any of three reasons. First, there may be a hardware malfunction. Second, the two sources of error of Theorem 3 may combine to generate an above-average error. (As we have seen, this should only occur to a small fraction of the clocks being synchronized.) Finally, in an improperly set-up machine, a series of high-priority interrupts may prevent the operating system from servicing lower-priority timer clock interrupts, causing that machine's clock to slow down. Given that TEMPO was designed for an environment where Byzantine faults are highly improbable, the synchronization algorithm can tolerate $(N - 1)/2$ faults. However, it should be noted that the clock of the master, which is not considered more important than any other clock by the fault-tolerant averaging function, may cause the clock difference measurement algorithm to fail if it is "double-faced."

VI. Comparison with Related Work

Although TEMPO is a distributed program, it uses a centralized approach in directing the synchronization activities. Fault-tolerance is achieved by not giving a privileged role to the master's clock in the synchronization algorithm and by providing an election algorithm that elects a new master should the old one terminate. Our approach therefore contrasts with other existing algorithms that adopt a fully distributed approach to fault-tolerance.

It is difficult to compare the various clock synchronization algorithms because, as observed by Lamport and Melliar-Smith [8], different algorithms require different methods of reading clocks and each method generates a different error. In addition, the various authors describe the bounds on their algorithms using parameters not always easily convertible to those of our system of variables. However, in general, the errors in clock synchronization, as in Theorem 3, depend on the *uncertainty* in the elapsed time between the generation and the receipt of a message and on the time between synchronization rounds.

In the remainder of this section, in order to compare the bounds on the accuracy of different algorithms, we make the following three additional assumptions: 1) there are $N = 3F + 1$ machines, where F is the number of machines with faulty clocks; 2) the message delivery time is in the range $[\tau - \eta, \tau + \eta]$, where τ is the median delay time and η is the uncertainty; and 3) the transmission times between any two machines are equally distributed.

Lundelius and Lynch [10] describe an algorithm that executes in a series of rounds; each round is started when a clock reaches a certain predefined value. When this happens, a machine broadcasts that value to all other machines. Meanwhile, it collects within a particular bounded amount of time measured on its own clock, messages from other machines. Then, each machine computes the correction for its clock using a fault-tolerant averaging function. The bound analysis shows that clocks can be synchronized as closely as $4\eta + 4\rho T$, but the authors suggest that, with a slight modification of their algorithm, they can reduce the second term to $2\rho T$.

The algorithm designed by Halpern *et al.* [6] is also based on the periodic broadcasting of clock values. In their method however, a machine that receives a message with a value that its clock has not reached yet, updates the clock to that value and broadcast the corresponding message. This algorithm generates an error of $\tau + \eta + 2\rho T$.

The three algorithms introduced by Lamport and Melliar-Smith [7], CON, COM, and CSM, are based on broadcast as well and achieve the following accuracy, respectively $2N\eta + N\rho T$, $2(N + 1)\eta + \rho T$, and $(N + 17)/3\eta + \rho T$. Although Lamport and Melliar-Smith do not give the synchronization error in a form comparable to ours—they analyze how closely in real time clocks reach the same value whereas we measure how close clocks are at the same real time—the two quantities appear to be similar.

Cristian, Aghili, and Strong [2] propose an algorithm in which nonfaulty clocks periodically generate *synch waves* that are propagated to all nodes of a point-to-point network of diameter D. (The network diameter is defined as the maximum distance between any two nodes. Notice that in an Ethernet local area network $D = 1$.) Independently generated synch waves eventually merge into a *winning wave* that distributes the time of the *fastest* clock to all the nodes. If we identify their *maximum link delay* with $\tau + \eta$, clocks can be as close as $D(\tau + \eta)(1 + \rho)$ upon synchronization, but can drift apart as much as $D(\tau + \eta)(1 + \rho) + [\rho(2 + \rho)/(1 + \rho)]\,[T(1 + \rho) + (\tau + \eta)]$ during the interval T between two synch waves.

The algorithm presented by Srikanth and Toueg [12] is based on the assumption that a known upper bound exists on the time required for a message to be prepared by a process, sent to all processes, and processed by them. As in some other algorithms discussed above, the synchronization takes place in rounds, which in this case start at multiples of a specified interval. Each process broadcasts a resynchronization message, and after receiving $F + 1$ resynchronization messages (so that at least one of them is from a nonfaulty process) accepts the resynchronization

and resets its time to the correct multiple of the interval, plus a constant conveniently chosen to avoid setting the time backwards. The clever way in which clocks reset their local times makes the accuracy achieved *optimal*, i.e., as close to the real time as that of a single clock. Identifying with $\tau + \eta$ the maximum link delay, the time required for a message to reach all processes is $D(\tau + \eta)$, where D is the diameter of the network. The paper shows that at any time any two clocks will differ by no more than $D(\tau + \eta)[(1 + \rho)^3 + \rho(2 + \rho)/(1 + \rho)]$.

While it is true that most communication protocols are designed to provide an upper bound on the communication time, perhaps by abnormally terminating the transmission after a number of retries, it is also true that the resulting variance in the transmission times can be much larger than the average transmission time. A unique feature of our algorithm is that it can bound the round-trip time, despite the high variance in transmission times, by rejecting those measurements that do not satisfy the requirements of Theorem 1. In fact, under the assumptions introduced above, if we call T_m the minimum transmission time, we have

$$\tau - \eta = T_m, \quad \tau + \eta = T_M - T_m;$$

and

$$\tau = \frac{T_M}{2}, \quad \eta = \frac{T_M - 2T_m}{2}.$$

By comparing the expression for η with (2), we can rewrite the result of Theorem 3 as

$$4\eta + 2\rho T.$$

Although the formula for the accuracy of our algorithm is the same as the one for the algorithm of Lundelius and Lynch, our η is much lower than theirs. Using for the parameters the values we have introduced in the previous section, we obtain $\tau = 10$ milliseconds and $\eta = 5$ milliseconds. In the case of other algorithms, η is proportional to the standard deviation of the transmission times, which for the Ethernet can be rather large when messages collide. When clocks are synchronized—or almost synchronized—the simultaneous broadcasting of messages that occurs in the algorithms may cause numerous collisions, increasing both the median transmission time τ and the uncertainty η. Therefore in an Ethernet environment, we would expect that our algorithm achieve significantly better synchronization accuracy. In a non-Ethernet environment, for instance a ring or point-to-point network, we would still expect that η of the other algorithms would be larger than our η, though the difference between the two may be smaller.

Algorithms COM and CSM were developed in the framework of Byzantine clock synchronization and both require about N^{F+1} messages. The algorithms of Lundelius and Lynch, Halpern *et al.*, Cristian, Aghili, and Strong, that of Srikanth and Toueg, and algorithm CON require in the worst case about N^2 messages. TEMPO, in contrast with the other algorithms, employs for each synchronization round only a linear number of messages. However, unlike TEMPO, which needs an election mechanism to ensure that a new master be elected in case the current one crashes or the network partitions, those algorithms are inherently fault-tolerant. Our choice was motivated by the fact that in our computing environment the kind of faults that require the intervention of the election procedure are rare. We have followed a design principle [9] that calls for simplicity in the most common situations and confines complexity and high costs with unusual conditions.

VII. Conclusions

We have discussed the upper and lower bounds on the accuracy achieved by the clock synchronization algorithms of TEMPO, which is distributed with Berkeley UNIX 4.3BSD. TEMPO keeps the clocks of VAX computers in a local area network synchronized with an accuracy comparable to the resolution of single machine clocks. Comparison with other clock synchronization algorithms shows that TEMPO, in an environment with no Byzantine faults, may achieve better synchronization at a lower cost.

Acknowledgment

The authors would like to thank D. Ferrari and M. Karels for their valuable advice during the development and implementation of these algorithms. V. Rangan and the referees provided helpful comments on how to improve the paper.

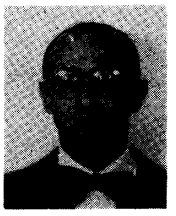
References

[1] "Adjtime System Call," *UNIX Programmer's Manual (Section 2)*, 4th Berkeley UNIX Distribution Release 3, Feb. 1985.
[2] F. Cristian, H. Aghili, and R. Strong, "Clock synchronization in the presence of omission and performance faults, and processors joins," in *Proc. 16th Annu. Symp. Fault-Tolerant Computing Systems*, Vienna, July 1986, pp. 218-223.
[3] R. Gusella and S. Zatti, "TEMPO—A network time controller for a distributed Berkeley UNIX system," *IEEE Distributed Processing Tech. Comm. Newslett.*, vol. 6, no. SI-2, pp. 7-15, June 1984.
[4] R. Gusella and S. Zatti, "The Berkeley UNIX 4.3BSD time synchronization protocol specification," Univ. California, Berkeley, Rep. UCB/CSD 85/250, June 1985.
[5] R. Gusella and S. Zatti, "An election algorithm for a distributed clock synchronization program," in *Proc. IEEE 6th Int. Conf. Distributed Computing Systems*, Boston, MA, May 1986, pp. 364-371.
[6] J. Halpern *et al.*, "Fault-tolerant clock synchronization," in *Proc. 3rd ACM Annu. Symp. Principles of Distributed Computing*, Vancouver, Aug. 1984, pp. 89-102.
[7] L. Lamport and P. M. Melliar-Smith, "Byzantine clock synchronization," in *Proc. 3rd ACM Annu. Symp. Principles of Distributed Computing*, Vancouver, Aug. 1984, pp. 68-74.
[8] ——, "Synchronizing clocks in the presence of faults," *J. ACM*, vol. 32, pp. 52-78, Jan. 1985.
[9] B. Lampson, "Hints for computer system design," in *Proc. 9th SOSP, Operating System Review, ACM*, vol. 17, Oct. 1983, pp. 33-48.
[10] J. Lundelius and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," in *Proc. 3rd ACM Annu. Symp. Principles of Distributed Computing*, Vancouver, Aug. 1984, pp. 75-88.
[11] J. Postel, Ed., "Internet control message protocol—DARPA Internet program protocol specification," USC/Inform. Sci. Inst., Rep. RFC 792, Sept. 1981.
[12] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J. ACM*, vol. 34, no. 3, pp. 626-645, July 1987.

Riccardo Gusella (S'79) received the Laurea degree (cum laude) in electrical engineering from the University of Padua, Italy, and the M.S. degree in computer science from the University of California, Berkeley.

He is currently completing the Ph.D. degree in computer science at the University of California, Berkeley, where he is a researcher in the Computer Systems Research Group. He has worked for the Italian Telephone Company (SIP) and for the Centro Studi e Laboratori Telecomunicazioni (CSELT) in Turin, Italy. During 1987 he was with the Computing Science Research Center of Bell Laboratories, Muray Hill, NJ. His research interests include distributed operating systems, the impact of communication technology on protocols and networking software, and performance analysis of communication systems. His thesis work is on modeling the arrival stream of packets generated by workstations on local area networks.

Dr. Gusella is a member of the Association for Computing Machinery. He has been a recipient of a Fulbright scholarship.



Stefano Zatti (S'83–M'85) received the Laurea degree (cum laude) in mathematics from the University of Pavia, Italy, in 1980, and the M.S. degree in computer science from the University of California, Berkeley, in 1985.

In 1982–1983 he worked for Olivetti in Ivrea, Italy, modeling the performance of Olivetti distributed operating systems. From 1983 to 1985 he was a member of the Computer Systems Research Group in Berkeley, CA, working on distributed clock synchronization and load balancing, and contributing to the release 4.3 of Berkeley UNIX. Since 1985 he has worked in the area of communications for IBM Research at the Zurich Laboratory, with particular emphasis on naming and directory services. His research interests include operating systems, distributed systems, computer network security, and computer performance analysis and modeling.

Dr. Zatti is a member of the Association for Computing Machinery and AICA.

Multisystem Coupling by a Combination of Data Sharing and Data Partitioning

JOEL L. WOLF, DANIEL M. DIAS, MEMBER, IEEE, BALAKRISHNA R. IYER, AND PHILIP S. YU, SENIOR MEMBER, IEEE

Abstract—**In a multisystem partitioned database system, the databases are partitioned among the multiple systems and a facility is provided to support the shipping of database requests among the systems. In contrast, in the data sharing multisystem approach, all systems have direct access to the shared database. There are a number of tradeoffs between these two approaches. In this paper we propose and evaluate a hybrid architecture that combines the approaches, and offers the advantages of each. Some databases are shared between systems, while others are retained private by specific systems. The issue is to determine which databases to share, which to retain private, and how to route transactions and partition the private databases among systems so as to minimize response time or overheads, while balancing the load among systems. A simulated annealing heuristic is used to solve this optimization problem. Trace data from large mainframe systems running IBM's IMS database management system are used to illustrate the methodology and to demonstrate the advantages of the hybrid approach.**

Index Terms—**Data sharing, function shipping, multiprocessor systems, simulated annealing, transaction processing.**

I. Introduction

THE large growth rate of the demand for computing capacity has made the coupling of multiple systems important. One method of locally coupling multiple systems for transaction processing is to partition the databases among the multiple systems and provide a facility to support the shipping of function requests among systems [1], [6]. Another approach to local multisystem coupling is the data sharing approach in which a number of systems, each running an independent operating system, share a common database at the disk level [10], [15]. There are a number of tradeoffs between the two approaches. In this paper we examine a hybrid approach, combining the data partitioning and data sharing approaches, and we attempt to provide the best features of each.

In the hybrid approach some databases are shared between the systems, while others are retained private by one of the systems. Hence it is necessary to determine which databases to share and which to retain private. This decision is based on the overheads involved in the two approaches and on the access patterns to the databases. In addition, transactions must be routed among the systems, so as to minimize remote calls to private databases which are not locally owned, while balancing the load among the systems. This optimization problem is solved by a heuristic based on simulated annealing [7], [8]. Traces from large mainframe systems running IBM's IMS database management system are used to examine the efficacy of the hybrid approach.

Section II outlines the data partitioning, data sharing and the hybrid approaches to multisystem coupling. The approaches are compared qualitatively. The optimization problem is formulated in Section III, and the simulated annealing approach to solving it is outlined. A quantitative comparison of the approaches is presented in Section IV. It is demonstrated that the hybrid approach has significant advantages for realistic workloads and system parameters. Finally, concluding remarks appear in Section V.

Manuscript received June 30, 1987.

The authors are with IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.

IEEE Log Number 8928288.

II. Hybrid Architecture and Qualitative Comparison

In this section we outline the tradeoffs between the data partitioning and data sharing approaches that motivate the hybrid architecture. The approaches are then compared qualitatively.

In the data partitioning approach, illustrated in Fig. 1, the databases are partitioned among multiple systems and can only be accessed directly by the owning system. Function request shipping is the capability for transactions executing in one database system to send database requests to other database systems (cohorts) for service. With such a facility, as in IBM's Customer Information Control System [5], [6], a transaction executing on any system can issue function requests referring to any database without detailed knowledge of how the databases are partitioned among systems. However, due to communication costs, performance degrades as the fraction of remote function requests increases. To optimize performance, transactions must be routed to systems so as to minimize remote function requests, while balancing the workload among systems. A two-phase commit protocol [4] is typically introduced to coordinate the updates across multiple systems. At termination, a transaction will cause two sets of exchanges to take place between the transaction and all of its cohorts. A failure to complete this processing in any cohort will cause all updates to be backed out.

0098-5589/89/0700-0854$01.00 © 1989 IEEE