Fault-Tolerant Clock Synchronization for Distributed Systems Using Continuous Synchronization Messages *

Alan Olson, Kang G. Shin

Real-Time Computing Laboratory
The University of Michigan
Ann Arbor, Michigan 48109-2122
{alan,kgshin}@eecs.umich.edu
313-763-0391 (voice); 313-763-4617 (fax)

Bruno J. Jambor

Martin Marietta Astronautics Group
P.O. Box 179, Denver, CO 80201
bjambor@falcon.den.mmc.com
303-977-1972 (voice); 303-977-1145 (fax)

Abstract

We present a probabilistic synchronization algorithm which sends periodic synchronization messages, instead of periodic *bursts* of synchronization messages as other algorithms do. Our "continuous" approach therefore avoids the burst network loads of other algorithms. Nodes always have current estimates of other nodes' clocks, allowing them to monitor the state of system synchronization, and adjust their clocks as needed. The algorithm is fault-tolerant, and may be easily adapted to a wide variety of systems and networks. We analyze and simulate the algorithm's performance on a 64-node hypercube, and show the algorithm provides tight synchronization while imposing only a light load on the network.

1 Introduction

The nodes of a distributed system often need to be *synchronized*. That is, there exists some δ, such that at any instant any two non-faulty nodes agree on the current time to within δ. This condition must persist even in the presence of faulty nodes, clocks, communications, etc. Synchronization is necessary to establish a global ordering of events. Real-time systems also use this global time base to allow each node to determine both when it must complete its own tasks, and when other nodes must complete theirs.

Synchronization may be accomplished by having all nodes use the same external time source. For example, Coordinated Universal Time (UTC) can be read via telephone, radio, or satellite, from several sources, at varying levels of accuracy [2, 17]. Alternatively, the system may be equipped with a single central clock and communications equipment to allow each node to read it. However, should this single time source fail, or if contact is lost or interrupted, the system is left without a clock. Also, external time sources may not be available for some applications (e.g., space vehicles.) And, there may be unacceptable cost, size, weight, and power consumption penalties imposed by radio/satellite receivers, communications equipment, etc.

It is often preferable to allow each node to use its own clock, and to limit differences between them with a synchronization algorithm. Synchronization algorithms may be classified according to the methods used by nodes to inform one another of their current clock values. *Hardware* algorithms [6, 7, 14, 16] use a dedicated network to broadcast each clock signal. *Network* algorithms [1, 3, 5, 8, 9, 11–13, 15] send messages across the existing communications network. Hardware algorithms provide tight synchronization, but are expensive to implement. The extra number of communications lines needed is on the order of n^2 for an n-node system. Network algorithms require no additional hardware, but tightness of synchronization is limited by uncertainty in communication delay. They also place an additional load on the communications network.

Probabilistic synchronization algorithms [1, 3, 4, 11, 13] are a type of network algorithm which try to compensate for the uncertainty in communications delay. The result is much tighter synchronization, but at the cost of an even greater load on the communications network. Worse, the load tends to be "bursty" — very high when the system is synchronizing, and zero otherwise. A master/slave clock organization may be used to reduce the network load [1, 3], but this creates problems with selection and synchronization of masters, faulty masters, and the additional load imposed on masters. Messages may be combined for efficiency [11, 13], but this creates long messages and may not significantly reduce the total number of bytes transmitted.

In this paper we propose a probabilistic synchronization

*The work reported in this paper was supported in part by Martin Marietta Astronautics Group in Denver, the Office of Naval Research under Grant N000-91-J-1115, the NASA under Grant NAG-1220, and the NSF under Grant MIP-9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

algorithm which provides tight synchronization while maintaining a constant, light network load. We do this by making our synchronization algorithm run *continuously*, instead of periodically as other algorithms do. Continuous operation eliminates burst loads, and each node is always aware of how far in or out of synchronization it is, so it can act to adjust its clock before problems arise. Another continuous synchronization algorithm is described in [4], but it is targeted towards completely connected systems while our algorithm is geared towards point-to-point systems. Our algorithm also allows for greater flexibility in adjusting the clock.

The rest of this paper is organized as follows. In Section 2 we give an overview of our algorithm and present our definitions and assumptions about the system. In Section 3 we describe the messages used to distribute clock information, and where and when they are sent. In Section 4 we discuss how each node keeps track of estimated skews, and updates these estimates in response to new information. In Section 5 we discuss how a node uses its skew estimates to adjust the value of its own clock. In Section 6 we present the results of our analysis and simulation. The paper concludes with Section 7.

2 Synchronization

We assume the system has n nodes, denoted N_0, ..., N_{n-1}. The synchronization algorithm must ensure that the difference (or *skew*) between the clocks of any two non-faulty nodes is never more than a given δ. The behavior of clocks at faulty nodes is not constrained.

Synchronization algorithms typically have three phases: distribution of clock information, estimation of clock skews, and adjustment of clock values. In conventional synchronization algorithms these phases are sequential, and the algorithm terminates after clock adjustment. In our algorithm all three phases are running simultaneously, and the algorithm never terminates. A node may therefore adjust its clock at any time, and other nodes must distinguish between clock adjustments and variation in clock rates. We do this with the following clock structure:

Raw clock: A counter that increments (approximately) at a specified rate. The raw clock has a specified *maximum drift*, ϱ. A raw clock with a specified rate of t ticks per second will actually have between $t(1-\varrho)$ and $t(1+\varrho)$ ticks per second. This property allows the raw clock to be used to measure time intervals with a known maximum error.

Clock adjustment: The value that is added to the raw clock value to get the current time. This allows the clock value to be adjusted without altering the raw clock.

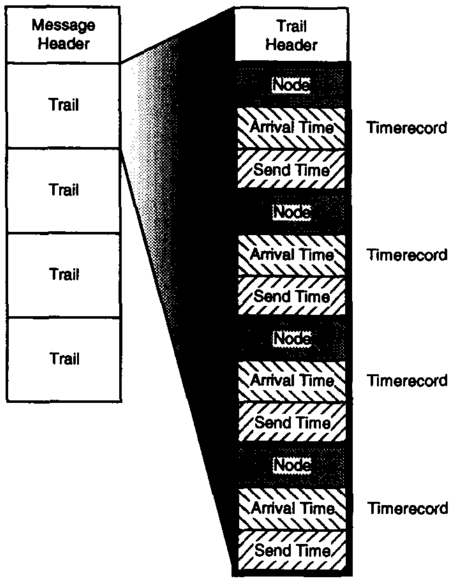

Figure 1: A synchronization message

Target adjustment: The value the clock adjustment *should have*. The synchronization algorithm does not change the clock adjustment immediately. Instead, it sets the target adjustment to the desired new value, and changes the clock adjustment slowly until it is equal to the target adjustment.

The synchronization algorithm sends both the raw clocks and target adjustments. Nodes may therefore estimate differences between raw clock values, and account for clock adjustment by using the most recent target adjustment.

3 Distribution of Clock Information

Our synchronization messages are novel both in their structure, and in the schedule for sending them.

3.1 Synchronization Message Structure

The basic unit of our synchronization message is the *trail*, which consists of one or more *timerecords*. The structure of synchronization messages is illustrated in Figure 1.

Each trail contains a header and a number of timerecords, listed in the order in which they were added. Nodes add a timerecord to each trail as it arrives. Timerecords are augmented timestamps, and each contains the following fields:

Node: The node which created the timerecord.

Arrival Time: The local raw clock when the trail arrived at the node which created the timerecord.

Send Time: The local raw clock and target adjustment when the trail was sent from the node which created the timerecord. The two values are recorded separately instead of being added together.

A trail is therefore a record of which nodes it has visited, and how long it stayed at each one. Should a trail return to the node where it started (i.e., it completes a cycle), this information can be used to estimate when it was at each node. This allows a node to estimate the clock skews between itself and the other nodes the trail visited.

Accurate recording of the arrival and send times on each timerecord is vital to the operation of the synchronization algorithm. Sloppiness in recording these times effectively increases uncertainty in message delays. Automatic timestamping schemes, like the one in [12], can greatly reduce uncertainty in message delays, and allow for tighter synchronization.

3.2 Synchronization Message Processing

The goal of the clock distribution algorithm is therefore to generate cyclic trails at each node. A flexible procedure for generating cyclic trails is described in [10]. Space restrictions prevent us from describing this procedure here. Instead, we describe a simpler method that works well in most cases.

The nodes of the system are laid out in a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ grid, as if the system were a wrapped square mesh. Dummy nodes are used to fill in the grid if necessary, and a system node is assigned to act in place of each dummy. Message sending is done in two alternating *slices*. During a *row* slice all nodes are sending synchronization messages to all the nodes in their row. Row slices are alternated with *column* slices; where all nodes are sending synchronization messages to all the nodes in their column. The length of a slice is a fixed system parameter, but must be significantly greater than 2δ.

Each synchronization message received is broken up into its component trails. Trails which have completed a cycle are sent to the estimation algorithm. The rest are stored temporarily for possible inclusion in one or more of the synchronization messages being sent in the next slice.

The trails sent in a synchronization message have two sources. Each synchronization message contains one new trail created just before the message was sent. The remaining trails are copies of trails received in the previous slice with the current node's timerecord added on. A trail is only included in a given synchronization message if doing so enables the trail to complete a cycle of length four. The rules for determining which trails go where are simple: a trail which arrives with one timerecord may be sent anywhere in the next slice; a trail which arrives with two timerecords may be sent only to the node in the same row or column as the node where it started; a trail which arrives with three timerecords must be sent back to the node where it started. The alternating nature of the slices ensure that each trail alternates between traveling "vertically" and "horizontally", and the resulting cycles are rectangles in the grid. These rules may be used to construct tables which allow each node to quickly determine suitable destinations for each trail.

4 Estimation

The estimation algorithm uses cyclic trails to estimate clock skews. We use an algorithm similar in concept to those in [3, 11], but adapted to continuous operation.

4.1 Skew Intervals

A node makes estimates by consulting the *skew intervals* maintained for each node. The bounds of N_i's skew interval for N_j are computed from N_j's timerecords on cyclic trails, and are guaranteed to bound the actual skew between N_j and N_i. The skew estimate is the midpoint of the interval, so the estimate can be off by no more than one-half of the width of the interval.

No constant interval is likely to contain the skew between a pair of nodes for long. Skew intervals must therefore be *widened* each time they are used. Widening is done by subtracting from the lower bound, and adding to the upper bound, the maximum clock drift possible since the last time the interval was widened.

N_i's skew interval for N_j is actually the intersection of a number of *trail intervals*. N_i computes a new trail interval for N_j each time it receives a cyclic trail containing a timerecord from N_j. The trail interval is then intersected with the existing (widened) skew interval to form the new skew interval.

Computing trail intervals is simple. Assume N_i has received a cyclic trail. Our distribution algorithm generates cycles of length four, so let the other three nodes the trail visited be N_{i_1}, N_{i_2}, and N_{i_3}. We need the following definitions:

T_j: The raw send time on the timerecord of N_{i_j}. T_0 is the raw send time on N_i's first timerecord. T_4 is the raw *arrival* time on N_i's second timerecord.

$\underset{j\to j+1}{d}$: The *minimum* delay between the sending of the synchronization message by N_{i_j} and its reception at $N_{i_{j+1}}$. These may also be summed: $\underset{j\to k}{d} \equiv \sum_{h=j}^{k} \underset{h\to h+1}{d}$.

W_j: The *measured wait* time at N_{i_j}. This is the difference between the raw send and arrival times at N_{i_j}. Note that W_0 and W_4 are both 0. The sum of a series of waits is also defined: $\underset{j\to k}{W} \equiv \sum_{h=j+1}^{k} W_h$.

When the trail returns to N_i, the raw clock at N_{i_j} must have a value of at least $T_j + \underset{j\to 4}{d} + \underset{j\to 4}{W}$. Similarly, the raw clock at N_{i_j} must have a value of at most $T_j + (T_4 - T_0) -$

$$\left[T_j - T_4 + \left(\underset{j\to 4}{d} + \frac{\underset{j\to 4}{W}}{1+\varrho}\right)(1-\varrho)\,,\ T_j - T_4 + \left(\frac{T_4 - T_0}{1-\varrho} - \underset{0\to j}{d} - \frac{\underset{0\to j}{W}}{1+\varrho}\right)(1+\varrho)\right] \tag{4.1}$$

$\underset{0\to j}{d} - \underset{0\to j}{W}$. These bounds only hold if there is no clock drift. The maximum clock drift, ϱ, is used to compensate. Also, T_4 is subtracted from each bound to convert them to bounds on the skew between N_{i_j} and N_i. The final interval for the skew between N_{i_j} and N_i is shown in Equation (4.1).

4.2 Analysis

The width of any given interval, and the uncertainty of the resulting estimate, is somewhat variable. Because the uncertainty of the estimates determines how closely nodes may be synchronized, we would like to be able to predict how wide any given interval is likely to be.

As above, we assume N_i receives a cyclic trail containing timerecords from N_{i_1}, N_{i_2}, and N_{i_3}. During the following slice N_i receives a similar cyclic trail, only the order of the timerecords is reversed: N_{i_3}, N_{i_2}, and N_{i_1}. In the next two slices the pattern repeats. We call trails with the N_{i_1}, N_{i_2}, N_{i_3} order *forward* trails, and the trails with the reverse order *backward* trails. These forward-backward pairs of trails form the basis of our analysis.

The skew interval for N_{i_j} is the intersection of all previously-computed trail intervals (appropriately widened to account for possible clock drift since they were computed.) In older intervals the widening is so great they are of no consequence in the final intersection. For this reason we consider only the most recent $2q$ trail intervals, where q has been chosen appropriately large.

Number the trails from 0 to $2q-1$, with the oldest trail being trail 0. We define the following notation for forward trails (arrow directions are reversed for backward trails):

$\alpha_{j\wr i}$: The skew between the raw clock values of N_{i_j} and N_i at the time N_i received the trail.

T_j^h: The timerecord of N_{i_j} from the h'th trail.

$\underset{j\to j+1}{\mathcal{X}^h}$: A random variable representing the difference between the actual delay and $\underset{j\to j+1}{d}$ for trail h (i.e., the difference between actual and minimum message delay.) The $\underset{j\to j+1}{\mathcal{X}}$'s are assumed independent and identically distributed. A series of these may be summed as for $\underset{j\to k}{d}$.

$\underset{j\to k}{\mathcal{W}^h}$: Similar to $\underset{j\to k}{W}$, except $\underset{j\to k}{\mathcal{W}^h}$ is a random variable representing the *actual* (instead of measured) node wait on trail h.

The following inequalities hold for all forward trails (there are analogous ones for backward trails):

$$T_4^h \ \geq\ T_0^h + \left(\underset{0\to 4}{d} + \underset{0\to 4}{\mathcal{X}^h} + \underset{0\to 4}{\mathcal{W}^h}\right)(1-\varrho) \tag{4.2}$$

$$T_4^h \ \leq\ T_0^h + \left(\underset{0\to 4}{d} + \underset{0\to 4}{\mathcal{X}^h} + \underset{0\to 4}{\mathcal{W}^h}\right)(1+\varrho) \tag{4.3}$$

$$T_j^h \ \geq\ T_0^h + \alpha_{j\wr i} + \left(\underset{0\to j}{d} + \underset{0\to j}{\mathcal{X}^h} + \underset{0\to j}{\mathcal{W}^h}\right)(1-\varrho) \tag{4.4}$$

$$T_j^h \ \leq\ T_0^h + \alpha_{j\wr i} + \left(\underset{0\to j}{d} + \underset{0\to j}{\mathcal{X}^h} + \underset{0\to j}{\mathcal{W}^h}\right)(1+\varrho) \tag{4.5}$$

Let L_j^h and U_j^h be the lower and upper bounds of the interval for N_{i_j} from trail h. We re-write the endpoints of the interval in Equation (4.1) in terms of our random variables, substituting from Equations (4.2) through (4.5) as needed. We also use the fact that ϱ is small to make the approximations $(1+\varrho)(1-2\varrho) \approx (1-\varrho)$ and $(1-\varrho)(1+2\varrho) \approx (1+\varrho)$. Let λ be the length of a slice. For forward trails the following inequalities hold (again, there are corresponding inequalities for backward trails):

$$\begin{aligned} L_j^h \ &\geq\ \alpha_{j\wr i} - \underset{j\to 4}{\mathcal{X}^h}(1+\varrho) - 2\varrho\lambda(2q-h-1) \\ &\quad - 2\varrho\left(\underset{0\to 4}{d} + \underset{0\to j}{\mathcal{X}^h} + \underset{0\to j}{\mathcal{W}^h} + 2\underset{j\to 4}{\mathcal{W}^h}\right) \end{aligned} \tag{4.6}$$

$$\begin{aligned} U_j^h \ &\leq\ \alpha_{j\wr i} + \underset{0\to j}{\mathcal{X}^h}(1+\varrho) + 2\varrho\lambda(2q-h-1) \\ &\quad + 2\varrho\left(2\underset{0\to 4}{d} + 2\underset{0\to 4}{\mathcal{X}^h} + 3\underset{0\to j}{\mathcal{W}^h} + 2\underset{j\to 4}{\mathcal{W}^h}\right) \end{aligned} \tag{4.7}$$

We eliminate dependency on node wait by replacing the final terms in Equations (4.6) and (4.7) with $2\varrho\beta^f_{L_i}$ and $2\varrho\beta^f_{U_i}$. $\beta^f_{L_i}$ and $\beta^f_{U_i}$ are chosen large enough to be highly probable upper bounds for the terms they replace. $\beta^b_{L_i}$ and $\beta^b_{U_i}$ are the corresponding values for backward trails.

Let $\mathcal{L}_j^p$ and $\mathcal{U}_j^p$ be the lower and upper bounds of the interval for N_{i_j} formed by intersecting the intervals of the p'th pair of forward-backward trails. Given the distribution functions for $\underset{0\to j}{\mathcal{X}}$, $\underset{j\to 4}{\mathcal{X}}$, $\underset{0\leftarrow j}{\mathcal{X}}$, and $\underset{j\leftarrow 4}{\mathcal{X}}$, the distribution functions for $\mathcal{L}_j^p$ and $\mathcal{U}_j^p$ are shown in Equations (4.8) and (4.9). The $\alpha_{j\wr i}$ is dropped from Equations (4.8) and (4.9) since it merely shifted both distributions the same amount in the same direction, and it is the *difference* between the random variables that is important.

The intersection of the intervals of the q forward-backward trail pairs has a lower bound equal to the maximum of the $\mathcal{L}_j^p$'s, and an upper bound equal to the minimum of the $\mathcal{U}_j^p$'s. We define two new random variables: MAX_j^{2q} is the maximum of the $\mathcal{L}_j^p$'s, and MIN_j^{2q} is the minimum of

$$F_{\mathcal{L}_j^p}(x) = \left(1 - F_{\underset{j\to 4}{\mathcal{X}}}\left(\frac{-x - 2\varrho\beta^f_{L_i} - 2\varrho\lambda(2q-2p-1)}{1+\varrho}\right)\right)\left(1 - F_{\underset{0\leftarrow j}{\mathcal{X}}}\left(\frac{-x - 2\varrho\beta^b_{L_i} - 2\varrho\lambda(2q-2p-2)}{1+\varrho}\right)\right) \quad (4.8)$$

$$F_{\mathcal{U}_j^p}(x) = 1 - \left(1 - F_{\underset{0\to j}{\mathcal{X}}}\left(\frac{x - 2\varrho\beta^f_{U_i} - 2\varrho\lambda(2q-2p-1)}{1+\varrho}\right)\right)\left(1 - F_{\underset{j\leftarrow 4}{\mathcal{X}}}\left(\frac{x - 2\varrho\beta^b_{U_i} - 2\varrho\lambda(2q-2p-2)}{1+\varrho}\right)\right) \quad (4.9)$$

the $\mathcal{U}_j^p$'s. The distribution functions for MAX_j^{2q} and MIN_j^{2q} can be expressed in terms of the distribution functions of $\mathcal{L}_j^p$ and $\mathcal{U}_j^p$:

$$F_{MAX_j^{2q}}(x) = \prod_{p=0}^{q-1} F_{\mathcal{L}_j^p}(x) \quad (4.10)$$

$$F_{MIN_j^{2q}}(x) = 1 - \prod_{p=0}^{q-1}(1 - F_{\mathcal{U}_j^p}(x)) \quad (4.11)$$

The density functions, $f_{MAX_j^{2q}}(x)$ and $f_{MIN_j^{2q}}(x)$, may be found by differentiating Equations (4.10) and (4.11).

Let Ω_j^{2q} be the width of N_i's interval for N_{i_j}. The probability that Ω_j^{2q} is less than x is the probability that the difference between MIN_j^{2q} and MAX_j^{2q} is less than x. Random variable MIN_j^{2q} depends on the values of $\mathcal{U}_j^p$. Each $\mathcal{U}_i^p$ depends on the values of $\underset{0\to j}{\mathcal{X}}^{2p}$, and $\underset{j\leftarrow 4}{\mathcal{X}}^{2p+1}$. Similarly, MAX_j^{2q} ultimately depends on the values of $\underset{j\to 4}{\mathcal{X}}^{2p}$ and $\underset{0\leftarrow j}{\mathcal{X}}^{2p+1}$. Therefore, MIN_j^{2q} and MAX_j^{2q} depend on different, independent, random variables. The difference between them can thus be computed by a simple convolution integral. With a little simplification we get:

$$F_{\Omega_j^{2q}}(x) \geq \int_{-x}^{0} f_{MAX_j^{2q}}(y) F_{MIN_j^{2q}}(x+y)\,dy \quad (4.12)$$

Section 6 contains examples using Equation (4.12), and compares them with simulation results.

4.3 Fault-Tolerance

Since all the information used to make estimates is carried on the trails, trails provide the only avenue through which a faulty node may affect estimates made by other non-faulty nodes. There are two ways a faulty node may proceed. A *transmission* fault causes a node to alter or destroy trails that it receives. An *accounting* fault causes a node to add misleading information of its own to the trail, but leave the rest of the trail untouched.

4.3.1 Transmission Faults. Transmission faults fall into two general categories: those that cause trails to be lost or sent to the incorrect destination, and those that alter or remove the timerecords of other nodes. If digital signatures are used to detect changes in other node's timerecords, then altered trails are essentially equivalent to lost trails.

Lost trails are detected almost immediately since each node knows what trails it should receive during each slice. Transmission faults are usually easy to find, and can often be masked with standard techniques like replication of messages. Nodes may also "skip" those nodes which act suspicious. For example, N_{i_1} sends the trail it received from N_i to both N_{i_2} and N_{i_3}. N_{i_3} uses the copy received from N_{i_1} if N_{i_2} does not forward the trail properly.

4.3.2 Accounting Faults. An accounting fault causes a node to record an incorrect arrival or send time on its timerecord. As a result, other nodes either under or over-estimate the wait time at the affected node. This affects the computation of intervals, and may cause actual skews to be outside the computed intervals.

There is no guaranteed method for detecting accounting faults. However, with a simplified, but fairly realistic, model of accounting faults we can get an idea of how likely they are to cause problems. Let N_i receive a series of cyclic trails containing timerecords from N_{i_1}, N_{i_2}, and N_{i_3}. Let N_{i_1} have an accounting fault which causes it to add *offsets* o_f and o_b to its measured wait time on forward and backward trails. This nicely models the most likely type of accounting fault: a clock which runs too fast or too slow. We wish to see the effects on N_i's intervals for N_{i_2} and N_{i_3}, so we modify Equation (4.8) by adding o_b to the numerator in $F_{\underset{0\leftarrow j}{\mathcal{X}}}$, and modify Equation (4.9) by adding o_f to the numerator in $F_{\underset{0\to j}{\mathcal{X}}}$.

Distribution functions for the new values of MAX_j^{2q} and MIN_j^{2q} may be found exactly as in Section 4.2. However, the width of the resulting interval is not the primary interest here. Instead, the following values are computed:

$P_{E_j^{2q}}$: The probability that the resulting interval is *empty*, i.e., the lower bound is greater than the upper bound. For an empty interval $MIN_j^{2q} - MAX_j^{2q} < 0$, and the density function is similar to Equation (4.12):

$P_{C_j^{2q}}$: The probability that the actual skew is within the resulting interval. If $F'_{MAX_j^{2q}}$ and $F'_{MIN_j^{2q}}$ are the modified density functions described above then

$$P_{C_j^{2q}} = F'_{MAX_j^{2q}}(0)\left(1 - F'_{MIN_j^{2q}}(0)\right) \quad (4.13)$$

Section 6.3 computes $P_{E_j^{2q}}$ and $P_{C_j^{2q}}$ for a 64-node hypercube.

5 Adjustment

Nodes use their skew estimates to compute a *synchronization adjustment.* The synchronization adjustment is then *applied* by adding it to the target adjustment. The unique features of our algorithm allows considerable flexibility in clock adjustment.

5.1 Computing the Synchronization Adjustment

A number of methods of computing synchronization adjustments have been proposed. For example, given the set of all skew estimates, and if no more than m nodes are faulty, one may choose the mean [8], $m+1$st smallest [12], or median (after the m smallest and largest estimates are discarded) [9]. Any of these may be used with our synchronization algorithm. However, in this section we present a variant of the interactive convergence algorithm [8] which has several unique features that make it especially well suited for our purposes.

Our algorithm is called *restricted range mean.* It begins with the set of all available skew estimates (including a 0 estimate for the local node), and finds the largest subset having the following properties:

1. The average uncertainties of the estimates is less than or equal to a specified $\varepsilon < \delta$.
2. For each estimate, the absolute value of the estimate, minus its uncertainty, is less than or equal to δ (i.e., the estimate's interval must intersect $[-\delta, \delta]$.)
3. For any two estimates, the absolute value of their difference, minus the sum of their uncertainties, is less than or equal to δ (i.e., the estimates must be within δ of one another.)

The members of this set are called the *accepted* estimates. If the number of accepted estimates is at least κ, the computation is *successful,* and the synchronization adjustment is the mean of the accepted estimates. The values of ε and κ depend both on one another, and on δ, n, and m. The relationship between these values is derived in Section 5.2.

Dropping property 3 produces a variant called *unrestricted range mean.* Unrestricted range mean has a larger value for κ, but it is easier to find the set of accepted estimates.

5.2 Proof of Correctness

Correctness proofs for traditional synchronization algorithms usually proceed by showing that when two nodes adjust their clocks simultaneously, the maximum skew between their clocks is τ, where $\tau < \delta$. The difference between τ and δ allows for clock drift between resynchronizations. These proofs depend on the assumption that all nodes adjust their clocks simultaneously. However, forcing simultaneous clock adjustments seems to violate the "spirit" of our algorithm.

We can use a similar proof if we assume that any pair of nodes successfully computes synchronization adjustments simultaneously (or nearly so.) This is a reasonable assumption if nodes compute synchronization adjustments at the end of each slice, and if computation of the synchronization adjustment is nearly always successful. We do not require nodes apply each successfully computed synchronization adjustment. Instead, nodes only apply synchronization adjustments that are greater than $\ell = \frac{1}{2}(\delta - \tau) - \varrho\lambda$. We know that if two nodes adjust their clocks simultaneously they will be within τ of one another. It follows that if a node doesn't adjust its clock it must be within $\tau + \ell$ of any node that did adjust its clock. Finally, we conclude that if two nodes don't adjust their clocks they must be within $\tau + 2\ell = \delta - 2\varrho\lambda$ of one another. Therefore, whether they adjust their clocks or not, any pair of non-faulty nodes will still be within δ of one another at the end of the next slice.

Let N_i and N_j be any two nodes in the system. Without loss of generality, assume N_j's clock is ahead of N_i's clock. Let N_i accept $e_i \geq \kappa$ estimates, and N_j accept $e_j \geq \kappa$ estimates. We wish to find, for all possible values of e_i and e_j, the maximum skew between N_j and N_i after they have adjusted their clocks. The maximum skew increases as e_i and e_j decrease, so κ is the smallest integer such that whenever both e_i and e_j are greater than or equal to κ, the maximum skew is less than or equal to τ.

The skew between N_j and N_i after they have adjusted their clocks is equal to the skew between them before they adjusted their clocks, plus the difference between the synchronization adjustments of N_j and N_i. There are certain conditions that *must* hold for the skew between N_j and N_i after they have adjusted their clocks to be the maximum for the given values of e_i and e_j:

1. N_i's estimates are, on average, ε too low, while N_j's estimates are, on average, ε too high.
2. Every non-faulty node is accepted by either N_i, N_j, or both.
3. Non-faulty nodes which are only accepted by N_j have skews of δ with respect to non-faulty nodes which are only accepted by N_i.

4. The number of faulty nodes is m, and *both* N_i and N_j accept every faulty node.

5. N_i's estimates for the faulty nodes are the smallest estimates N_i can accept, and N_j's skew estimates for these same faulty nodes are the largest estimates N_j can accept (recall that estimates of faulty nodes don't have to agree.)

The following definitions are useful in the computation of κ. Note that in these definitions all skews are the *actual* skews instead of the estimated skews. The difference between actual and estimated skew is accounted for by including a term for maximum uncertainty.

$\underline{\gamma_i}$ and $\overline{\gamma_i}$: The minimum and maximum skews (with respect to N_i) of non-faulty nodes accepted by N_i. Note that a node automatically accepts itself, so $\underline{\gamma_i}$ has a maximum of 0, and $\overline{\gamma_i}$ has a minimum of 0. Also, $\overline{\gamma_i} - \underline{\gamma_i} \le \delta$.

$\underline{\gamma_j}$ and $\overline{\gamma_j}$: The minimum and maximum skews (with respect to N_j) of non-faulty nodes accepted by N_j. Again, the maximum of $\underline{\gamma_j}$ is 0, the minimum of $\overline{\gamma_j}$ is 0, and $\overline{\gamma_j} - \underline{\gamma_j} \le \delta$.

Γ_i: The sum of the skews (with respect to N_i) of the $n - e_j$ non-faulty nodes which are only accepted by N_i.

Γ_j: The sum of the skews (with respect to N_j) of the $n - e_i$ non-faulty nodes which are only accepted by N_j.

Γ_{ij}: The sum of the skews (with respect to N_i) of the $e_i + e_j - n - m$ non-faulty nodes accepted by *both* N_i and N_j.

N_i estimates skews of $\overline{\gamma_i} - \delta$ for each faulty node, and N_j estimates skews of $\underline{\gamma_j} + \delta$ for each faulty node.

$$\begin{aligned}
\tau \;&>\; \alpha_{j\wr i} + \frac{1}{e_j}\left(\Gamma_{ij} - (e_i + e_j - n - m)\,\alpha_{j\wr i}\right.\\
&\qquad \left. + \Gamma_j + \left(\underline{\gamma_j} + \delta\right) m + e_j\varepsilon\right)\\
&\qquad - \frac{1}{e_i}\left(\Gamma_{ij} + \Gamma_i + (\overline{\gamma_i} - \delta)\, m - e_i\varepsilon\right)\\
e_i e_j \tau \;&>\; (e_i - e_j)\,\Gamma_{ij} + e_i\Gamma_j - e_j\Gamma_i\\
&\qquad - e_i\,(e_i - n - m)\,\alpha_{j\wr i} + e_i m \underline{\gamma_j} - e_j m \overline{\gamma_i}\\
&\qquad + (e_i + e_j)\, m\delta + 2 e_i e_j \varepsilon \qquad (5.1)
\end{aligned}$$

To eliminate Γ_i and Γ_j from Equation (5.1), we replace them with the following bounds:

$$\begin{aligned}
\Gamma_i \;&\ge\; \underline{\gamma_i}\,(n - e_j)\\
\Gamma_j \;&\le\; \left(\underline{\gamma_i} + \delta - \alpha_{j\wr i}\right)(n - e_i)
\end{aligned}$$

Since N_i and N_j both accept estimates of some common nodes, it follows that $\underline{\gamma_j} + \alpha_{j\wr i} \le \overline{\gamma_i}$. Substituting into Equation (5.1) gives:

$$\begin{aligned}
e_i e_j \tau \;&>\; (e_i - e_j)\,\Gamma_{ij} + 2 e_i e_j \varepsilon - e_i\,(e_i - n)\,\underline{\gamma_i}\\
&\qquad + e_j\,(e_j - n)\,\overline{\gamma_i} + (e_i - e_j)\, m \overline{\gamma_i}\\
&\qquad - e_i\,(e_i - n)\,\delta + (e_i + e_j)\, m\delta \qquad (5.2)
\end{aligned}$$

No further progress can be made without additional information about the values of e_i and e_j. Consider first the case where $e_i \ge e_j$. It follows that Γ_{ij} has its maximum value, $\left(\underline{\gamma_i} + \delta\right)(e_i + e_j - n - m)$, and that $\overline{\gamma_i}$ has its maximum value, $\underline{\gamma_i} + \delta$. Substituting into Equation (5.2) gives:

$$\tau \;>\; \frac{1}{e_j} m\delta + \frac{1}{e_i}\,(n + m - e_j)\,\delta + 2\varepsilon$$

Both e_i and e_j divide positive quantities, so the right-hand side is maximized when both are equal to κ. Substituting and solving for κ gives:

$$\kappa \;>\; \delta\,(n + 2m)\,/\,(\delta + \tau - 2\varepsilon) \qquad (5.3)$$

The case where $e_i \le e_j$ proceeds similarly, and the result is identical to Equation (5.3).

A similar derivation may be done for unrestricted range mean. The result is nearly identical to Equation (5.3):

$$\kappa \;>\; \delta\,(n + 3m)\,/\,(\delta + \tau - 2\varepsilon) \qquad (5.4)$$

Ensuring successful computation of the synchronization adjustment requires manipulating the relative values of δ, τ, m, and ε.

6 Analysis and Simulation

The adjustment algorithm of Section 5 has requirements of the estimation algorithm of Section 4. We use both the analysis of Section 4 and simulation to show that these requirements can be met for realistic values of δ and λ. We also use the analysis of Section 4.3.2 to analyze the effects of accounting faults.

6.1 Parameters

Both the analysis and simulation assume a store-and-forward type message passing system, where all delays are independent and identically distributed. We assume each delay to be exponentially distributed with a minimum of 2110μs and a mean of 2450μs. That is, if the distance between N_{i_j} and $N_{i_{j+1}}$ is k, then $\underset{j \to j+1}{d}$ is $k(2110)\mu$s, and $\underset{j \to j+1}{x}$ is the sum of k independent exponentially-distributed random variables with mean 340μs. These delays are similar to those used in [1, 3], and are rather large by modern standards; they are about what one would expect for user-level processes on workstations connected by Ethernet.

A value of 10^{-5} is used for ϱ. The simulator chooses the clock rate for each node at random from the range $[1 - \varrho, 1 + \varrho]$. The clock rates remain fixed throughout the simulation.

6.2 64-node Hypercube

For this example we use a 64-node hypercube. When we map the nodes of the hypercube onto an 8×8 square mesh, we find that each row and column corresponds to a 3-dimensional subcube. Each synchronization message encounters at most three message delays, and any cyclic trail encounters at most twelve.

6.2.1 Analysis. We consider the width of the intervals computed by N_i. We concentrate on the 49 nodes which do not share a row or column with N_i. These nodes are the "middle" nodes in the trails, and are always N_{i_2} in the analysis. We concentrate on these nodes because their intervals are usually the widest. This is because these nodes tend to be further away, and because N_i receives more cyclic trails containing the timerecords of nodes with which it shares a row or column.

Our assumption of independent, identically distributed message delays means the distributions $F_{\underset{2\rightarrow 4}{\mathcal{X}}}$, $F_{\underset{0\leftarrow 2}{\mathcal{X}}}$, $F_{\underset{0\rightarrow 2}{\mathcal{X}}}$, and $F_{\underset{2\leftarrow 4}{\mathcal{X}}}$ are identical. Wait times are on the order of λ, and since λ is much larger than $\underset{j\rightarrow j+1}{d}$ and $\underset{j\rightarrow j+1}{\mathcal{X}}$, the β values are dominated by the wait times. We therefore use the approximations $\beta^f_{\mathcal{L}_2} = \beta^b_{\mathcal{L}_2} \approx= 4\lambda$, and $\beta^f_{\mathcal{U}_2} = \beta^b_{\mathcal{U}_2} \approx 8\lambda$. Table 1 shows how the probability the width of an interval is less than 2ε varies with the number of hops from N_i. The first column shows the number of hops, and the second column shows how many nodes are that many hops from N_i. The last four columns show the probability, for several values of ε and λ, that the intervals of these nodes have widths less than 2ε.

One can use results like those in Table 1 to compute the number of *automatic* intervals; those intervals with a width less than 2ε. Automatic intervals produce estimates with uncertainties less than ε, and which will therefore be automatically accepted when computing the synchronization adjustment. N_i's interval for itself has width 0 and is therefore automatic. Our experience shows the intervals for the 15 nodes in N_i's row and column are also automatic. We can use the analytical results to predict how many of the remaining 49 intervals will be automatic.

For example, when $\varepsilon = 1000\mu$s and $\lambda = 1$s, we expect that of the 49 nodes in Table 1, $9 + 18 + 15(0.9976) + 6(0.8206) + 1(0.3035) = 47.19$ will have automatic intervals. The expected number of automatic intervals is therefore 63.19. If we use range restriction, $\delta = 5000\mu$s, and $\tau = 4000\mu$s, then Equation (5.3) gives $\kappa > 62.86$ when $m = 12$. The expected number of automatic intervals is therefore enough to ensure successful computation of the synchronization adjustment when up to 12 faults are present.

6.2.2 Simulation. Because automatic intervals do not provide the only accepted estimates, considering only automatic intervals produces pessimistic results. We turn to simulation both to determine the expected number of accepted estimates, and the expected maximum skew. Table 2 shows the number of automatic and accepted intervals in our simulation. If we compare the number of automatic intervals from Table 2 with the expected number of automatic intervals from Section 6.2.1, we find the simulation results to be consistently higher. The differences are considerable, from 37 automatic intervals to 52, in one instance.

Figure 2 plots the maximum skew over time for different values of λ. The maximum skew is fairly constant, and well below δ. Letting $\delta = 5$ms may not seem impressive at first glance, but it must be seen in light of the relatively large message delays. Minimal message delay between the two furthest nodes is over 12ms. And, if we assume the maximum message delay is a mere 2ms greater than minimum (5ms might be a more reasonable estimate), the maximum variability in message delay is 12ms. Our algorithm therefore does much better than synchronization algorithms such as [8, 9, 15], which are limited by the maximum variability in message delays.

A final result of the simulation is an estimate of the size of synchronization messages. Assuming 64-bit clock values, the messages were just under 12Kbytes long, and length was independent of λ. Each node sends 7 synchronization messages per slice, which take a total of 12 hops, resulting in 144Kbytes transmitted. If $\lambda = 4$s, this works out to about 12Kbytes per link per second, or less than 1% of the available bandwidth of a 10Mbit/s link. The load is not perfectly constant, but can be made nearly so if synchronization messages are broken up into small pieces. These pieces may be sent individually over the course of the entire slice without adversely affecting synchronization.

6.3 Accounting Faults

For our final example we use the results of Section 4.3.2 to analyze accounting faults in a 64-node hypercube. As before, we assume that N_i receives a series of cyclic trails where N_{i_1} has an accounting fault, and consider the effects on N_i's intervals for N_{i_2} and N_{i_3}. N_{i_1} expresses the accounting fault by adding offsets o_f and o_b to its wait times on forward and backward trails. This causes N_i's intervals for N_{i_2} and N_{i_3} to either become empty, or no longer contain the actual skew. An empty interval shows that an accounting fault is present. An interval which does not contain the actual skew invalidates the guaranteed uncertainty provided by the estimation algorithm.

Our goal, therefore, is to show that N_i's intervals for N_{i_2} and N_{i_3} will either be empty, or will contain the actual

		$\lambda = 1$s		$\lambda = 4$s	
hops	nodes	$\varepsilon = 1000\mu$s	$\varepsilon = 2000\mu$s	$\varepsilon = 1000\mu$s	$\varepsilon = 2000\mu$s
2	9	1.0000	1.0000	0.9139	1.0000
3	18	1.0000	1.0000	0.3812	1.0000
4	15	0.9976	1.0000	0.0588	1.0000
5	6	0.8206	1.0000	0.0048	0.9953
6	1	0.3035	1.0000	0.0003	0.8939

Table 1: Probability interval widths are less than 2ε in a 64-node hypercube.

	$\varepsilon = 1000\mu$s		$\varepsilon = 2000\mu$s	
λ (s)	automatic	accepted	automatic	accepted
1	63.2	64.0	64.0	64.0
2	60.6	64.0	64.0	64.0
3	56.8	64.0	64.0	64.0
4	52.6	64.0	64.0	64.0

Table 2: Average number of automatic and accepted intervals in a 64-node hypercube.

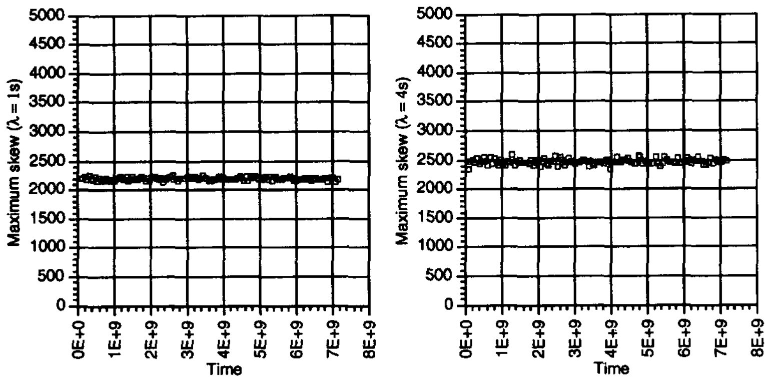

Figure 2: Maximum skew between nodes in a 64-node hypercube when $\delta = 5000\mu$s, $\tau = 4000\mu$s and $\varepsilon = 1000\mu$s.

forward hops	backward hops	$\lambda = 1$s			$\lambda = 4$s		
		$P_{E_j^{2q}}$	$P_{C_j^{2q}}$	$P_{F_j^{2q}}$	$P_{E_j^{2q}}$	$P_{C_j^{2q}}$	$P_{F_j^{2q}}$
2	2	1.0000	0.0000	0.0000	0.6117	0.1059	0.2824
3	1	0.9994	0.0001	0.0005	0.0605	0.5863	0.3532
4	4	0.9261	0.0227	0.0512	0.0159	0.7689	0.2152
6	2	0.1010	0.6698	0.2292	0.0000	0.9881	0.0119
7	1	0.0247	0.8883	0.0870	0.0000	0.9973	0.0027
6	6	0.1123	0.6370	0.2507	0.0001	0.9776	0.0223
9	3	0.0001	0.9922	0.0077	0.0000	0.9999	0.0001

Table 3: Analysis of accounting faults for a 64-node hypercube where $o_f = o_b = 1000\mu$s.

skew. Table 3 shows $P_{E_j^{2q}}$ (the probability the interval is empty) and $P_{C_j^{2q}}$ (the probability the interval contains the actual skew) for several possible cyclic trails in the hypercube. The first two columns show the number of hops to the node of interest (either N_{i_2} or N_{i_3}) in forward and backward trails. Note that the forward and backward hops are equal for N_{i_2}, and not equal for N_{i_3}. For example, N_0 receives a series of trails which travel in one direction or the other along the path $N_0 \leftrightarrow N_7 \leftrightarrow N_{15} \leftrightarrow N_8 \leftrightarrow N_0$. In this case, N_{15} is N_{0_2} and the number of hops to it is 4 for either forward or backward trails, while N_8 is N_{0_3} and the number of hops to it is 7 for forward trails and 1 for backward trails. The remaining columns show $P_{E_j^{2q}}$, $P_{C_j^{2q}}$, and $P_{F_j^{2q}} \equiv 1 - P_{E_j^{2q}} - P_{C_j^{2q}}$ for various values of λ.

An examination of Table 3 shows that $P_{E_j^{2q}}$ decreases and $P_{C_j^{2q}}$ increases both with the number of hops, and with λ. N_{i_3} has a lower value of $P_{E_j^{2q}}$ and a higher value of $P_{C_j^{2q}}$ than N_{i_2}, and the differences increase with the number of forward hops. Clearly, distance and time ameliorate the effects of accounting faults. $P_{F_j^{2q}}$, the probability that N_i is unaware of the fault and is using an incorrect estimate, first rises, then falls as the number of hops increases. The accounting fault therefore has a limited window of opportunity. The error it can introduce into estimates is no larger than either o_f or o_b, but larger offsets increase the likelihood of detection, and smaller offsets have little hope of causing any errors at all.

7 Conclusion

Clock synchronization is a requirement of many real-time distributed systems. In this paper we presented a probabilistic fault-tolerant clock synchronization algorithm which can be used with a variety of network architectures. This algorithm is very different from existing synchronization algorithms, and has a number of unique features:

- Synchronization messages are sent at short, regular intervals, instead the periodic burst of messages sent by other synchronization algorithms.
- An average of estimate uncertainties is used instead of a limit on estimate uncertainty.
- Nodes may adjust their clocks at any time.

We presented an analysis of our algorithm, and also did simulation. Both showed the algorithm to work quite well on 64-node hypercubes.

References

[1] K. Arvind, "Probabilistic clock synchronization in distributed systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 5, pp. 474–487, May 1994.

[2] R. E. Beehler and D. W. Allan, "Recent trends in NBS time and frequency distribution services," *Proceedings of the IEEE*, vol. 74, no. 1, pp. 155–157, January 1986.

[3] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 4, no. 3, pp. 146–158, 1989.

[4] F. Cristian and C. Fetzer, "Probabilistic internal clock synchronization," in *Proc. 13th Symposium on Reliable Distributed Systems*, pp. 22–31, Dana Point, CA, 1994.

[5] J. Y. Halpern, B. Simons, R. Strong, and D. Dolev, "Fault-tolerant clock synchronization," in *Proc. 3rd Symp. on Principles of Distributed Computing*, pp. 89–102, 1984.

[6] J. L. W. Kessels, "Two designs of a fault-tolerant clocking system," *IEEE Trans. Computers*, vol. C-33, no. 10, pp. 912–919, October 1984.

[7] C. M. Krishna, K. G. Shin, and R. W. Butler, "Ensuring fault tolerance of phase-locked clocks," *IEEE Trans. Computers*, vol. C-34, no. 8, pp. 752–756, August 1985.

[8] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of the ACM*, vol. 32, no. 1, pp. 52–78, January 1985.

[9] J. Lundelius-Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1–36, 1988.

[10] A. Olson, *Synchronization of Fault-Tolerant Distributed Real-Time Multicomputers*, PhD thesis, The University of Michigan, 1994.

[11] A. Olson and K. G. Shin, "Probabilistic clock synchronization in large distributed systems," in *Proceedings of the 11th Intl. Conference on Distributed Computing Systems*, pp. 290–297. IEEE, May 1991.

[12] P. Ramanathan, D. D. Kandlur, and K. G. Shin, "Hardware assisted software clock synchronization for homogeneous distributed systems," *IEEE Trans. Computers*, vol. C-39, no. 4, pp. 514–524, April 1990.

[13] S. Rangarajan and S. K. Tripathi, "Efficient synchronization of clocks in a distributed system," in *Proc. Real-Time Systems Symposium*, pp. 22–31, December 1991.

[14] K. G. Shin and P. Ramanathan, "Clock synchronization of a large multiprocessor system in the presence of malicious faults," *IEEE Trans. Computers*, vol. C-36, no. 1, pp. 2–12, January 1987.

[15] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *Journal of the ACM*, vol. 34, no. 3, pp. 626–645, July 1987.

[16] N. Vasanthavada and P. N. Marinos, "Synchronization of fault-tolerant clocks in the presence of malicious failures," *IEEE Trans. Computers*, vol. 37, no. 4, pp. 440–448, April 1988.

[17] G. M. R. Winkler, "Changes at USNO in global timekeeping," *Proceedings of the IEEE*, vol. 74, no. 1, pp. 151–155, January 1986.