PROBABILISTIC CLOCK SYNCHRONIZATION IN LARGE DISTRIBUTED SYSTEMS

Alan Olson Kang G. Shin

Real–Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109–2122.

ABSTRACT

Probabilistic clock synchronization algorithms can provide tight synchronization without the use of special hardware. However, all probabilistic algorithms proposed to date require a master/slave organization of clocks, and can require a large number of synchronization messages. These two characteristics can make them unsuitable for use in large distributed systems. In this paper we propose a synchronization algorithm which uses one of two probabilistic techniques to estimate remote clock values, and uses an interactive convergence algorithm on the resulting estimates to adjust the local clock. The algorithm does not require master/slave clocks and reduces the number of messages needed. As a result it is suitable for use in large distributed systems.

1 Introduction

Clock synchronization in a distributed system is a basic requirement of many applications. In [8] it is shown how a common time base can be used to decrease the number of messages which must be exchanged to insure proper rollback recovery. In real-time systems the concept of time is central to system operation, and a deadline is meaningless if different parts of the system have greatly differing ideas of the current time.

Clock synchronization in a distributed system is a problem which has been studied extensively in recent years. A number of different synchronization schemes have been proposed [1, 2, 3, 4, 5, 6, 7, 9, 10, 11]. These algorithms can generally be classified as either hardware, software, or probabilistic. Hardware schemes [3, 4, 9, 11] provide tight synchronization through the use of special hardware. Software schemes [5, 6, 10] use special synchronization messages to synchronize clocks and thus require no special hardware, but do not provide tight synchronization. Probabilistic schemes [1, 2] provide tight synchronization and use no special hardware, but require greater numbers of synchronization messages.

To date, all probabilistic clock synchronization algorithms have used a master/slave clock organization. This is largely because probabilistic algorithms are simply estimation techniques, which provide highly accurate estimates of a single remote clock at the expense of a large number of synchronization messages. This can cause problems when synchronizing a large system. A single master clock could be swamped by large numbers of synchronization messages, or the increased traffic load around the master could invalidate some of the assumptions made about communication delay times. Multiple masters could be used, but such a setup would require some method of synchronizing the masters. One could avoid the need for master clocks if each node obtained estimates of all other clocks in the system, but the number of messages needed would be prohibitive.

In this paper we propose a probabilistic synchronization algorithm which does not use a master/slave clock organization, but limits the number of synchronization messages. It is therefore suitable for large distributed systems. Our algorithm uses either and interval-oriented approach or an averaging approach to estimate the values of remote clocks, then feeds these values to an interactive convergence algorithm to adjust the local clock. We reduce the number of messages by using one message to provide estimates for all the nodes on the path which it travels, and by not requiring each node to estimate the clock of every other node in the system.

The work reported in this paper was supported in part by the NASA under Grant NAG-1-296, the ONR under Contract N00014-85-K-0122, and Martin Marietta, Aerospace Division under Contract No. GH9-118088. 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.

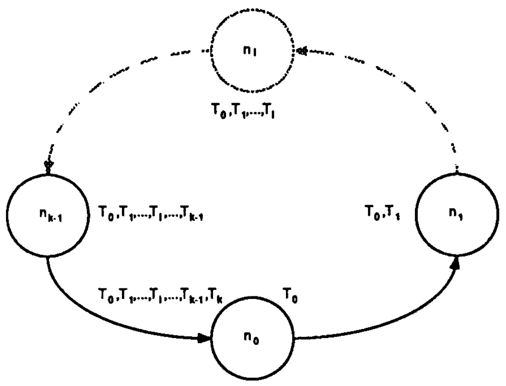

Figure 1: Path of synchronization message.

The rest of the paper is organized as follows. In Section 2 we present our estimation methods. In Section 3 we show how they can be used to synchronize a large system. The paper concludes with Section 4.

2 Two Estimation Methods

The first part of our synchronization algorithm is for each node to obtain estimates of the skew between its local clock and other clocks in the system. In this section we present two methods of obtaining these estimates. Both are probabilistic, they assume message delays can be modeled as independent random variables, and there is a non-zero probability that estimates will not have the desired accuracy. However, this probability can be made as small as is desired, given some information about the message delay distribution.

Both estimation methods use the same means of gathering the information needed to make estimates. Synchronization messages are sent along a cyclic path, and each node on the path adds its local time to the message. Consider Figure 1, a message is sent from n_0 at local time T_0, passes through $k-1$ nodes each of which adds its local time T_i, and returns to n_0 at local time T_k. Each message is sent around the path twice, so each node on the path will both send the message and later receive it with all timestamps attached.

2.1 An Interval-Oriented Approach

This algorithm is much like the one described in [2], each node uses the information in the message to compute an interval containing the current value of each remote clock. However, whereas in [2] the results of earlier trials are thrown away while the algorithm waits for a small enough interval, we keep the results of each trial and intersect them with the results of later trials. As a result, the synchronization process is speeded considerably.

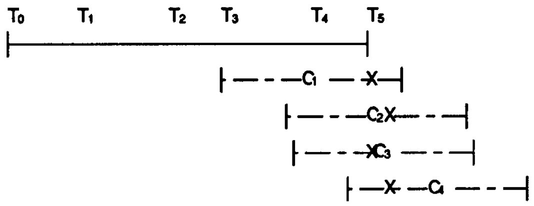

Figure 2: Intervals calculated from timestamps

2.1.1 Calculation of Intervals

Let ρ be the maximum clock drift rate, and d_{min} be the minimum time required to make one hop in the path. Let D be the total time taken by the message as measured on n_0's clock, i.e., $D = T_k - T_0$. Let D_A be the *actual* time taken for the message to make the trip around the cycle, and d_i be the difference between the actual time taken and the minimum time required for the message to be sent from n_i to n_{i+1}. Then

$$D_A = kd_{min} + \sum_{j=0}^{k-1} d_j .$$

Since the d_i's all have non-negative values, for $0 \leq i \leq k-1$ we get

$$0 \leq \sum_{j=i}^{k-1} d_j \leq D_A - kd_{min}. \tag{2.1}$$

For any $0 \leq i \leq k-1$, if $C_i(T_k)$ is the time on n_i's clock at time T_k on n_0's clock, then we get

$$C_i(T_k) \in \left[T_i + ((k-i)d_{min} + \sum_{j=i}^{k-1} d_j)(1-\rho), \right.$$
$$\left. T_i + ((k-i)d_{min} + \sum_{j=i}^{k-1} d_j)(1+\rho) \right] .$$

Substituting from Eq. (2.1) gives:

$$C_i(T_k) \in [T_i + (k-i)d_{min}(1-\rho),$$
$$T_i + (D_A - id_{min})(1+\rho)].$$

Noting that $D_A \leq D(1+\rho)$, and dropping the ρ^2 term, we finally get:

$$C_i(T_k) \in [T_i + (k-i)d_{min}(1-\rho),$$
$$T_i + D(1+2\rho) - id_{min}(1+\rho)]. \tag{2.2}$$

Figure 2 shows the results of a synchronization message sent along a 5-node path. Time increases to the right, the uppermost line shows the relative values of each of the timestamps on the message, and the four lower lines show the intervals which would be calculated for each of the $C_i(T_5)$'s. The corresponding skew intervals can be obtained by subtracting T_5 from each endpoint. If a number of messages are sent, a number of skew intervals will be obtained for each clock, and the actual skew will be in the intersection. Under the assumption of independent, random distribution of delay times, the intersection will tend to shrink as more messages are sent. This can be sped up by sending messages in both directions along the path since as can be seen from Figure 2, clocks early in the path have their intervals shifted left while later clocks have their intervals shifted right. In this way, the intervals of nearby nodes will shrink quickly while those of more distant nodes will take longer. We will refer to this process of shrinking intervals as *convergence*.

2.1.2 Analysis

Ideally, we want $F_{conv}(k,n)$, the probability that all intervals have converged after n messages have been sent given k nodes are on the path. In general, this function is difficult to determine. A more realistic goal is to calculate $F^{mid}_{conv}(k,n)$, the probability that the interval for the "middle" node in the path has converged after n messages have been sent given k nodes are on the path. As was noted above, more distant nodes on the path will take greater amounts of time for their intervals to converge, thus a high probability that the interval of the middle node has converged means a high probability that all intersections are small.

From Eq. (2.2) the bounds on a given interval are functions of two independent random variables: T_{head} the time the message took to get to the node, and T_{tail} the time the message took to return. If we assume clock drift to be negligible then $T_{head} + T_{tail} = D$. Also, if $T_{\lceil k/2\rceil}$ is measured relative to T_0 then $T_{\lceil k/2\rceil}$ is just $T_{head} + \alpha$ where α is the skew between clocks 0 and $\lceil k/2\rceil$. Eq. (2.2) now gives us:

$$\begin{aligned}C_{\lceil k/2\rceil}(T_k) \in [&T_{head} + \alpha + \lfloor k/2\rfloor d_{min}(1-\rho),\\ &2T_{head}(1+\rho) + \alpha + T_{tail}(1+2\rho)\\ &-\lceil k/2\rceil d_{min}(1+\rho)].\end{aligned}$$

We can obtain the skew interval by subtracting $D = T_{head} + T_{tail}$ from the endpoints of the above interval:

$$\begin{aligned}[&-T_{tail} + \alpha + \lfloor k/2\rfloor d_{min}(1-\rho),\\ &T_{head} + 2\rho D + \alpha - \lceil k/2\rceil d_{min}(1+\rho)].\end{aligned}$$

As ρ is typically small (usually on the order of 10^{-5} or less), we drop the $2\rho D$ term to get:

$$\begin{aligned}[&-T_{tail} + \alpha + \lfloor k/2\rfloor d_{min}(1-\rho),\\ &T_{head} + \alpha - \lceil k/2\rceil d_{min}(1+\rho)]. \qquad (2.3)\end{aligned}$$

A similar small ρ argument can be used to remove the remaining ρ's in Eq. (2.3), or in Eq. (2.2). Here our goal is to remove the D term from the upper bound of the interval so we can model the interval endpoints as independent random variables.

Let L_i and U_i be the lower and upper bounds of the skew interval generated by the i-th message. Given the distribution and density functions F_{head}, f_{head}, F_{tail}, and f_{tail}, of T_{head} and T_{tail}, we can obtain the distribution and density functions of L_i and U_i:

$$F_{L_i}(x) = F_{tail}(-x),\ f_{L_i}(x) = f_{tail}(-x)$$

$$F_{U_i}(x) = F_{head}(x),\ f_{U_i}(x) = f_{head}(x).$$

The constants have been ignored for the moment as they will be taken care of later. As these distribution functions are independent of i we will drop the subscript i. We want the maximum of the L_i's and the minimum of the U_i's:

$$\begin{aligned}MAX_n &= maximum\{L_i : 1 \le i \le n\}\\ MIN_n &= minimum\{U_i : 1 \le i \le n\}.\end{aligned}$$

The distribution functions for MAX and MIN can be expressed in terms of the distribution functions of L and U:

$$\begin{aligned}F_{MAX}(x;n) &= (F_L(x))^n\\ F_{MIN}(x;n) &= 1 - (1 - F_U(x))^n.\end{aligned}$$

The density functions are found by differentiating:

$$\begin{aligned}f_{MAX}(x;n) &= n(F_L(x))^{n-1} f_L(x)\\ f_{MIN}(x;n) &= n(1 - F_U(x))^{n-1} f_U(x).\end{aligned}$$

$F^{mid}_{conv}(k,n)$ is the probability that the width of the intersection is below a given constant ω. This probability is expressed in Eq. (2.5), where we have now added in the constants we ignored earlier. In the case of odd k an extra $-d_{min}\rho$ term is dropped from the left side of the inequality in Eq. (2.5). As MIN and MAX are independent, this probability can be computed by the convolution integral in Eq. (2.5).

2.1.3 Examples

We will assume k is even so that T_{head} and T_{tail} will have identical distributions. A normal distribution is a good choice for T_{head} and T_{tail}, especially as the number of nodes increases. This is because T_{head} and T_{tail} are the sums of $k/2$ (assumed) independent delays, the central limit theorem predicts the distribution of such a sum will tend towards normal as k increases. It further predicts that if the mean and variance of the delay for one hop on the path are μ and σ^2, then the mean and variance for T_{head} and T_{tail} will be $k\mu/2$ and $k\sigma^2/2$.

$$F^{mid}_{conv}(k,n) = P[(MIN_n + \alpha - \lceil k/2 \rceil d_{min}(1+\rho)) - (MAX_n + \alpha + \lfloor k/2 \rfloor d_{min}(1-\rho)) < \omega]$$

$$= P[MIN_n - MAX_n - kd_{min} < \omega] \quad (2.4)$$

$$= \int_{-\infty}^{\infty} dy \int_{-\infty}^{\omega+y+kd_{min}} f_{MAX}(y;n) f_{MIN}(x;n)dx. \quad (2.5)$$

k	$\omega/2$	n	F^{mid}_{conv}	where
16	1	32	0.5234	17.45
16	1	48	0.7218	23.76
16	1	96	0.9531	34.33
16	2	8	0.9160	3.64
16	2	16	0.9966	4.22
16	2	32	0.9999	4.26
32	1	1024	0.0472	638.80
32	1	8192	0.6554	4102.49
32	2	512	0.7393	244.44
32	2	2048	0.9992	385.42

Table 1: Normal distribution where $\sigma = 0.3$

k	$\omega/2$	n	F^{mid}_{conv}	where
16	1	8	0.9785	2.98
16	1	16	0.9998	3.13
16	1	24	0.9999	3.14
32	1	8	0.7611	4.32
32	1	16	0.9674	5.82
32	1	24	0.9960	6.19
32	1	32	0.9995	6.25
64	1	64	0.9830	19.49
64	1	96	0.9985	20.22

Table 2: Normal distribution where $\sigma = 1.0$

In [1, 2] the values of d_{min} and μ were given as 2.11 milliseconds and 2.45 milliseconds. [1] also gave σ as 1 millisecond, though we will also use 0.3 millisecond. We also consider several values of $\omega/2$, half the desired width of the interval and the maximum error if the middle of the interval is taken as the estimate.

Tables 1 and 2 show the value of F^{mid}_{conv} for various values of k, $\omega/2$, and n, each table is for a different value of σ. The *where* column shows, on average, how many messages had to be sent to achieve convergence. This information was produced by a simulator which was used to check the results of the analysis.

2.2 An Averaging Approach

If the average delay μ is known, then from the timestamp T_i of the remote node one should be able to estimate the current time on that node to be $T_i + (k-i)\mu$. This estimate is vulnerable to variation in the delivery times, but if the process is repeated often enough and the results averaged, the variation will likely average out so that the estimate obtained will be close to the true value. Such an algorithm cannot produce an interval which is *certain* to contain the correct value of the remote clock, but it does provide a probability that the true value of the clock is within a given distance from the estimate. This probability can be made to be as close to one as is desired.

We assume that messages are sent in both directions along the path and that n messages are sent in each direction, for a total of $2n$ messages. Each message will still go around twice, but nodes will only attach their times on the first trip, the second trip is so that nodes can see the times of nodes later on the path. To simplify notation we will assume that the messages alternate directions each time. If T^j_0 and T^j_k are the send and receipt times for message j, then we estimate the difference between n_i's clock and n_0's clock as

$$\alpha_{est} = \overline{T_k} - \overline{T_i} - k\mu/2, \quad (2.6)$$

where

$$\overline{T_k} = 1/2n \sum_{j=1}^{2n} T^j_k$$

$$\overline{T_i} = 1/2n \sum_{j=1}^{n} \left(T^{2j-1}_i + T^{2j}_{k-i}\right).$$

The X's in Figure 2 show the estimates which would be made for each clock on the path.

The skew value produced is only an estimate and may be arbitrarily far from the actual value. However, the laws of large numbers predict that the estimate will be close. Let the difference between the estimate and the actual value be ϵ. The maximum distance the estimate is allowed to vary from the true value is ϵ_{max}. For any given ϵ_{max} there is a finite probability that ϵ is greater than ϵ_{max}. This probability is called the *probability of invalidity*, and much of the next section will focus on how this probability can be made as small as possible.

2.2.1 Analysis

This averaging algorithm is similar to the one proposed in [1] and a detailed analysis of its operation can be found there. Our algorithm is different from that in [1], however, in that a group of processors is being synchronized and that there are two different variances in delivery times.

Eq. (2.6) can be re-written as

$$\alpha_{est} = \frac{1}{2n}\sum_{j=1}^{n}\left(\left(T_k^{2j-1} - T_i^{2j-1} - (k-i)\mu\right) + \left(T_k^{2j} - T_{k-i}^{2j} - i\mu\right)\right).$$

We would like to know the distribution of α_{est}. Since it is expressed as the sum of a number of independent random variables, we can use the central limit theorem to approximate it by a normal distribution. To find the appropriate normal distribution we need to know the mean and variance of each of the terms of the sum. The mean of each term should be α, the skew. The variance of each term will be the variance in time taken for the message to travel either $k-i$ hops or i hops. If we assume the time taken for each hop on the path to be independent, then we can assume the variances to be $(k-i)\sigma^2$ and $i\sigma^2$ where σ^2 is the variance of a single hop on the path. The distribution of α_{est} can thus be approximated by $\mathcal{N}(\alpha, \frac{k\sigma^2}{4n})$.

What is needed is a method of determining n given the maximum error ϵ_{max} and the desired probability that $\epsilon < \epsilon_{max}$. From the approximate distribution above we can determine the following:

$$P_{\epsilon<\epsilon_{max}} = Erf\left[\frac{\epsilon_{max}\sqrt{2n}}{\sqrt{k}\sigma}\right]$$

where Erf is the error function. Solving for n yields:

$$n = \frac{k\sigma^2}{2\epsilon_{max}^2} Erf^{-1}[P_{\epsilon<\epsilon_{max}}]^2. \tag{2.7}$$

2.2.2 Examples

With Eq. (2.7) it is easy to compute the number of messages which need to be sent. It is also easy to see the relationships between n and the various parameters σ, k, ϵ_{max}, and $P_{\epsilon<\epsilon_{max}}$. We have computed the values of n for several combinations of these parameters, and tabulated the results in Table 3 and 4.

k	ϵ_{max}	$P_{\epsilon<\epsilon_{max}}$	$2n$
16	1	0.9	2
16	1	0.999	8
16	1	0.99999	14
32	1	0.9	4
32	1	0.999	16
32	1	0.99999	28
64	1	0.9	8
64	1	0.999	32
64	1	0.999999	56

Table 3: Analysis for $\sigma = 0.3$

k	ϵ_{max}	$P_{\epsilon<\epsilon_{max}}$	$2n$
16	1	0.9	22
16	1	0.999	88
16	1	0.99999	156
32	1	0.9	44
32	1	0.999	174
32	1	0.99999	312
32	2	0.9	12
32	2	0.999	44
32	2	0.999999	80

Table 4: Analysis for $\sigma = 1.0$

3 Large Systems

If the system has a Hamiltonian cycle, one could use a single message path to synchronize the entire system. However, as path length increases, the accuracy of estimates declines. Also, a single faulty node could foul up the entire process. In this section we show how to avoid these problems.

Our solution is to use a divide-and-conquer approach. A number of synchronization groups are formed, so that each node belongs to at least one group. The groups are chosen so that there is a cycle which goes through each member, and one of the estimation algorithms is used to give each member an estimate of every other member's clock. When all groups have finished, each node takes the estimates it has collected and uses an interactive convergence algorithm to adjust its local clock. If the groups are set up properly all the clocks within a synchronization group will remain within a given skew, δ, of each other. The nodes which belong to more than one group can be used as "bridges" to provide a maximum skew between two nodes not in the same group, i.e., if A is within δ of B, and B is within δ of C, then A is within 2δ of C.

3.1 Tightness of Synchronization

To maintain synchronization within a group all the clocks in a group must remain within δ of one another.

To maintain synchronization between groups it is necessary for there to be some overlap between them. A node which belongs to a group of fast clocks and a group of slow clocks may find itself caught in the middle and unable to remain within δ of clocks from both groups. In this section we will describe the interactions between groups and the effect this has upon δ.

Definition 1: Two nodes A and B are said to be *grouped* if they belong to the same synchronization group. They are said to be *δ-synchronized* if their clocks always remain within δ of one another. The system is δ-synchronized if all pairs of grouped nodes are δ-synchronized.

To show δ-synchronization we must take a close look at how nodes are grouped. Assume A and B are grouped and the size of the synchronization group is k. Let G^A be the set of nodes with which A is grouped, G^B be the set of nodes with which B is grouped, and n_A and n_B be the number of nodes in each set. We then partition $G^A \cup G^B$ into the following seven sets.

The first set is $G_\cap^{AB} \triangleq G^A \cap G^B$, the nodes with which both A and B are grouped. Each of these nodes must be within δ of *both* A and B. Let the number of nodes in this set be $n_\cap$, since A and B are in the same synchronization group, $n_\cap \geq k-2$.

The next two sets are $G_1^A \subseteq G^A - G^B$ and $G_1^B \subseteq G^B - G^A$. These are nodes which, while not grouped with both A and B, are grouped with one of them and with a node which is grouped with the other. For example, if $a \in G_1^A$ then there exists some $b \in G_1^B$ such that a and b are grouped. We require that there be a bijection $g_1 : G_1^A \rightarrow G_1^B$ such that $g_1(a) = b \Rightarrow a$ and b are grouped. This pairing of nodes insures that $|G_1^A| = |G_1^B| = n_1$ and will have other important consequences later.

The next two sets are $G_2^A \subseteq G^A - G^B - G_1^A$, and $G_2^B \subseteq G^B - G^A - G_1^B$. These sets are much like G_1^A and G_1^B, only one more node has been added in the cycle. As an example, if $a \in G_2^A$ then there exists $b \in G_2^B$ and node c such that both a and b are grouped with c but not with each other. Again we require a bijection $g_2 : G_2^A \rightarrow G_2^B$ such that $g_2(a) = b \Rightarrow \exists c$ such that both a and b are grouped with c but not each other. We then have $|G_2^A| = |G_2^B| = n_2$.

The last two sets $G_u^A \triangleq G^A - G^B - G_1^A - G_2^A$, and $G_u^B \triangleq G^B - G^A - G_1^B - G_2^B$, contain whatever is left over. These nodes are those which are grouped with either A or B but have no useful relationship to the other which can be used to constrain the value of its clock. Denote the sizes of these sets by n_{A_u} and n_{B_u}.

Figure 3 attempts to provide some feel for the relationships between these sets, especially the maximum skew between two nodes of different sets. Nodes A and B are δ apart, while the nodes of $G_\cap^{AB}$, which must be within δ of both A and B, are shown in between them. Each node of G_1^A must be within δ of some node in G_1^B, so these sets are shown δ apart. The situation is similar for sets G_2^A and G_2^B, which must be 2δ apart. Finally, sets G_u^A and G_u^B, which are only constrained by their relationships with A and B, may be up to 3δ apart. The figure is oversimplified in the sense that nodes do not have to be exactly where they are pictured. For example, the nodes of G_1^A may be anywhere within δ of where they are pictured. But if a node in G_1^A is moved it drags with it its partner node in G_1^B. When the interactive convergence calculations are made by A and B what is added to one node is subtracted from the other. So the skew between the two nodes is no different from what it would be if the node was where it is pictured.

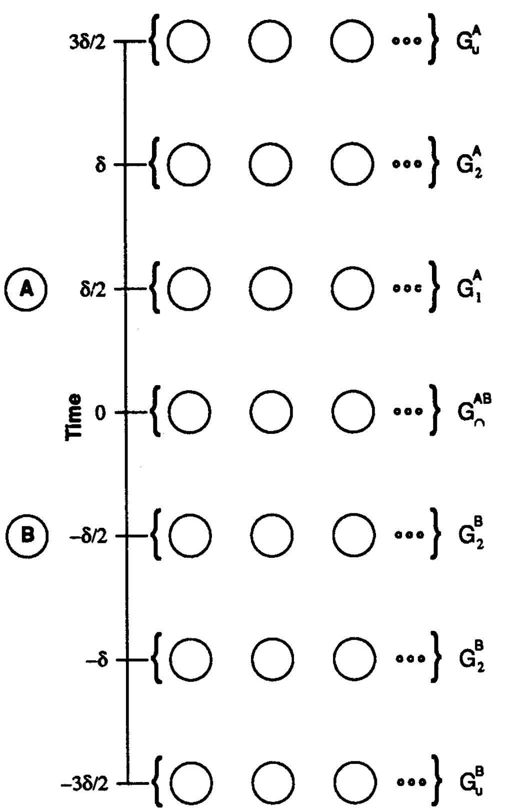

Figure 3: Partitioning of groupings

We can now calculate the difference in the clocks of A and B after synchronization. As shown in Figure 3, we assume the clock values of the nodes of $G_\cap^{AB}$ to be 0. We also assume that a node will get the least accurate reading possible of a remote clock, and the one which will do the most to make sure that A and B are not within δ after synchronization. Thus A will read the values of $G_\cap^{AB}$ to be ϵ while B sees them as $-\epsilon$. The maximum value of A's clock after synchronization C_{synch}^A is:

$$\begin{aligned} &C_{synch}^A \\ &\quad= (-\delta/2+\epsilon+n_\cap\epsilon+\delta/2+n_1(\delta/2+\epsilon) \\ &\qquad +n_2(\delta+\epsilon)+n_{A_*}(3\delta/2+\epsilon))/n_A \\ &\quad= \left(\frac{\delta}{2}(n_1+2n_2+3n_{A_*})+\epsilon(n_A-1)\right)/n_A. \end{aligned}$$

The situation for the minimum value of B is similar, only the signs are reversed and n_{B_*} and n_B are used. The difference between the two, *maxdiff* $= C_{synch}^A - C_{synch}^B$ is then:

$$\begin{aligned} maxdiff &= \frac{\frac{\delta}{2}(n_1+2n_2+3n_{A_*})+\epsilon(n_A-1)}{n_A} \\ &\quad+\frac{\frac{\delta}{2}(n_1+2n_2+3n_{B_*})+\epsilon(n_B-1)}{n_B}. \qquad (3.1) \end{aligned}$$

If, as is often the case, $n_A = n_B = n$, then $n_{A_*} = n_{B_*} = n_u$ and we can simplify Eq. (3.1) to:

$$maxdiff = 2\left(\frac{\delta}{2}(n_1+2n_2+3n_u)+\epsilon(n-1)\right)/n. \qquad (3.2)$$

These equations can be used to get the relationship between ϵ and δ. Once n_A, n_B, $n_\cap$, n_1, and n_2 are fixed, set $\delta >$ *maxdiff*, this can be solved to yield a minimum ratio of δ to ϵ. They can also be used to get the maximum time between synchronizations, τ_r. If ρ is the maximum clock drift rate, then $(\delta - maxdiff)/2\rho = \tau_r$.

3.2 Examples

We will investigate two different setups to synchronize a 256-node hypercube system. A hypercube has a natural structure for synchronization groups: the subcube. For both examples we will use 4-cubes (16 nodes), and will use asterisks (*) to designate the don't-care positions in the hypercube address.

3.2.1 A 32-group setup

For our first example we will use 32 subcubes as our synchronization groups, 16 of the form *abcd* * * * * where the upper four bits of the address are fixed, and 16 of the form * * * * *efgh* where the lower four bits of the address are fixed. Each node will be in 2 synchronization groups with 16 members each, thus we can use Eq. (3.2) with $n = 31$. If A and B are grouped then $G_\cap^{AB}$ is just the rest of the nodes in the synchronization group, thus $n_\cap = 14$. The remaining 15 nodes in each of G^A and G^B will be in G_1^A and G_1^B. This can be seen as follows: assume A has address 0000*abcd* and B has address 0000*efgh*, for any node *wxyzabcd* which is grouped with with A but not B, there is node *wxyzefgh* which is grouped with with B but not with A, *wxyzabcd* and *wxyzefgh* are both in the synchronization group *wxyz* * * * * and thus are grouped. This gives us $n_1 = 15$ and $n_2 = n_u = 0$, plugging this into Eq. (3.2) gives

$$\begin{aligned} \delta &> maxdiff = 2(15\delta/2+30\epsilon)/31 \\ &> \frac{15}{4}\epsilon. \end{aligned}$$

The value of δ must be at least $\frac{15}{4}$ times that of ϵ in order to ensure δ-synchronization.

With this setup, all nodes in the system will be within 2δ of one another. Let $\rho = 2$ microseconds per second, and $\epsilon = 1$ millisecond. If we want all nodes in the system to be within 10 milliseconds of one another we must have $\delta = 5$ milliseconds, then *maxdiff* $= 4.35$ milliseconds and a maximum time to resynchronization of $(5 - 4.35)/0.004 = 162.5$ seconds. If we are willing to let the maximum skew become 20 milliseconds we can have $\delta = 10$ milliseconds, *maxdiff* $= 6.77$ milliseconds, and a maximum time to resynchronization of $(10 - 6.77)/0.004 = 807.5$ seconds.

The number of synchronization messages sent depends on the standard deviation of delivery time and the synchronization algorithm used. If $\sigma = 1.0$, then we use the interval algorithm which takes at most 24 messages (but averages just over 3) to get an accuracy of 1 millisecond. Each message is sent twice by each node in the group and each node belongs to two groups, this means a maximum of 96 messages sent per node (and an average close to 12). If $\sigma = 0.3$, then we use the averaging algorithm which will send 14 messages per synchronization or a total of 56 messages per node.

3.2.2 A 17-group setup

As a second example, we reduce the number of synchronization groups. We keep the first 16, the subcubes of the form *abcd* * * * *, but take only the subcube * * * * 0000 from the second 16. The last subcube contains one node from each of the first 16 so, the subcubes are all "connected" in that any two nodes will be within 3δ of each other if δ-synchronization is maintained. With this division there are three possible cases for any two grouped nodes: they can both be in one of the first 16 subcubes only, one can be in the first 16 subcubes only while the other is in the last subcube, or both can be in the last subcube. Each case will place a different restriction on the ratio between δ and ϵ.

Suppose A and B are both in one of the first 16 subcubes, but neither is in the last subcube. Then $n_A = n_B$, $n_\cap = 14$, and we can use Eq. (3.2) to determine the ratio of δ and ϵ. In this case it turns out $\delta > \frac{15}{8}\epsilon$.

Suppose A is in one of the first 16 subcubes only, while B is also in the last subcube. In this case the two nodes are in different numbers of synchronization groups, A is in one while B is in two. For this case we must use Eq. (3.1). To start we note that $n_A = 16$ and $n_B = 31$. All the nodes in G^A (except A and B) will be in $G_\cap^{AB}$, so $n_\cap = 14$. The remaining nodes of G^B will be in G_u^B, so $n_{B_u} = 15$. We can now calculate

$$\begin{aligned} \delta &> 15\epsilon/16 + (45\delta/2 + 30\epsilon)/31 \\ &> \frac{945}{136}\epsilon. \end{aligned}$$

Finally, assume both A and B are in the last subcube. In this case we can once again use Eq. (3.2) as both nodes belong to two synchronization groups. We have $n = 31$, $n_\cap = 14$, and $n_u = 15$. From Eq. (3.2) we get

$$\begin{aligned} \delta &> 2(45\delta/2 + 30\epsilon)/31 \\ \delta &< -\frac{30}{7}\epsilon. \end{aligned}$$

As both δ and ϵ are positive values, this equation has no solutions. There are no values of δ and ϵ for which this setup can guarantee δ-synchronization.

4 Conclusion

In this paper we have presented a scheme for synchronizing the nodes of a large distributed system. Three major advantages of this scheme are : (i) it is software based and needs no special hardware support, (ii) high accuracy in estimation of clock values leads to tighter synchronization than is allowed by most software schemes, (iii) tightness of system synchronization can be controlled by the number of groups.

A further advantage of this approach is the ability to place co-operating tasks in the same synchronization group. This allows them to be closely synchronized without incurring the extra overhead involved in increasing the system synchronization.

As a matter of further research, we are considering the fault-tolerance of this scheme. A fault can only affect synchronization within a small area, and the more groups there are, the less affect a fault will have. We are currently considering methods of determining the fault-tolerance of a given synchronization setup.

References

[1] K. Arvind, "A new probabilistic algorithm for clock syncronization," in *Proc. Real-Time Systems Symposium*, pp. 330–339, Santa Monica, CA, 1989.

[2] F. Cristian, "Probalistic clock synchronization," *Distributed Computing*, vol. 3, pp. 146–158, 1989.

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

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

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

[6] 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.

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

[8] P. Ramanathan and K. G. Shin, "Checkpointing and rollback recovery using common time base," in *Proc. 7th IEEE Symp. on Reliable Distributed Systems*, pp. 13–21, October 1988.

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

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

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