A Global Time Reference for Hypercube Multicomputers

James C. French

IPC-TR-88-10
October 10, 1988

Institute for Parallel Computation
School of Engineering and Applied Science
University of Virginia
Charlottesville, VA 22903

This research was supported in part by JPL Contract #957721 and by the Department of Energy under Grant DE-FG05-88ER25063.

Abstract

This report describes a simple but effective technique for achieving a global time reference on hypercube multicomputers. The synchronization algorithm presented runs in $O(\log N)$ time where $N = 2^n$ is the number of processors in an order n hypercube. Empirical evidence is given to show that the expected clock agreement is $n / 2$ ticks. Since the synchronization algorithm does not depend on particular parameters of the underlying hardware, it is inherently portable.

A sampling technique for testing the agreement of the processor clocks in $O(N)$ time is also presented. This technique assumes that the internode message transit time, m, is constant but unknown.

One unexpected outcome of this work was the discovery that, in general, timing information cannot be reliably obtained on the NCUBE/AT system except perhaps in the total absence of message traffic.

1. Introduction

This report describes a simple but effective technique for achieving a global time reference on hypercube multicomputers. Citing variable message transmission times, queuing delays, and crystals running at different rates, Tanenbaum [TANE85] states that for large distributed systems "having a single global time is impossible." The situation is quite different when considering a tightly coupled network of homogeneous processors. For example, all the nodes of the NCUBE/ten hypercube receive their timing signal from a single crystal through the backplane. This implies that the clock ensemble (the processor clock on each node) will run at the same rate with a drift proportional to signal propagation delays. A reliable global time reference would be achieved if all processor clocks were started at the same instant with the same initial value. This is clearly not feasible. We can, however arrive at a reasonable global time reference by synchronizing the processor clocks. A very simple clock synchronization algorithm which provides a stable, high quality global time reference on the NCUBE/ten hypercube is described below.

In this report, we are concerned with the synchronization of a properly functioning set of processor clocks to form a globally distributed time service. The major issues involved with synchronizing clocks in a distributed system are covered in the seminal paper by Lamport [LAMP78]. Other researchers ([LAMP85, SHIN87, SRIK87, VASA88]) have considered clock synchronization in the presence of faults.

Examples of the utility of global time are abundant. A global time reference would be invaluable for many parallel simulation protocols. It can be used to generate unique identifiers as in C.mmp [JONE80] and timestamps in database applications. The ability to log asynchronous events and retain their temporal relationship is a necessary debugging aid for parallel programs. A global time reference can be useful in building system diagnostic software. A novel method for process synchronization using global time has also been proposed by Lamport [LAMP84] . It would also be useful for automatically scheduling checkpoints in long running jobs. This small

sampling of applications should be sufficient to justify some effort toward providing a global time reference on parallel machines.

The remainder of this paper is organized as follows. A brief description of hypercube multicomputers is given in the next section. The synchronization algorithm is described in section 3. Empirical results obtained from measurements on NCUBE hardware concerning clock stability and agreement are given in section 4. This section also presents an efficient clock sampling technique and discusses timing anomalies associated with the NCUBE/AT hypercube. Section 5 contains concluding remarks.

2. The Hypercube Multicomputer

The hypercube multicomputer is a message passing (as distinct from shared memory) MIMD computer. A hypercube multicomputer of order *n* is a collection of $N = 2^n$ homogeneous processors interconnected to form a boolean *n*-cube. The processors are customarily called *nodes* and are conceptually placed at the vertices of the *n*-cube. Thus, each node is directly connected to *n* neighbors, but may communicate with every other node. The nodes of the hypercube are labeled from 0 through $2^n - 1$ in such a way that the binary representations of adjacent node labels differ in exactly one bit. An example of the processor interconnection for an order 3 hypercube is shown in Figure 1.

The properties of the hypercube topology are well known and a good summary can be found in [SAAD88]. The regularity of the hypercube gives rise to many desirable properties. The hypercube has excellent connectivity with a maximum internode distance of *n* hops and has simple message routing and broadcast algorithms. Many other topological structures (e.g., rings, trees, meshes) can be embedded in the hypercube. It is this latter property that gives the hypercube much of its versatility and computational flexibility.

The work reported in this paper was done on NCUBE hardware, specifically the NCUBE/ten and NCUBE/AT hypercubes. The NCUBE/ten system used has 64 nodes each with 512 Kb of memory. The NCUBE/AT system uses an Intel 80386 based PC as a host processor

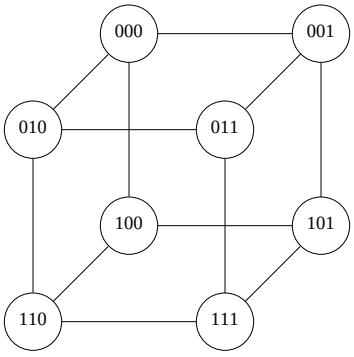

Figure 1: Order 3 Hypercube

with a 4 node hypercube processor array on a separate board. Each node of this NCUBE/AT system has 128 Kb of memory. The NCUBE/AT and the NCUBE/ten use the same node processor chips.

Both the NCUBE/ten and the NCUBE/AT run the AXIS operating system on their host processors. On the NCUBE/ten communication is done via VORTEX, the host I/O board node operating system, and VERTEX, the compute node operating system, while the NCUBE/AT communicates using VOERTEX which supports the combined functionality of VORTEX and VERTEX. More detail on the NCUBE hardware and software can be found in [HAYE86a, HAYE86b].

3. Synchronization Algorithm

Each node k is assumed to have a physical clock PC_k. For our purposes, this physical clock is the timing mechanism available to an application program (e.g., *ntime()* on the NCUBE systems); it is not the hardware clock or crystal generating the timing pulse for the processor, although it is derived from this timing source. The physical clock measures time in *ticks*, the duration of which is determined by the particular hardware at hand. The NCUBE/ten (NCUBE/AT) is clocked at 7 MHz (6 MHz) and increments its physical clock every 1024 pulses.[1] This equates to .146 msec (.171 msec) per tick.

Define a virtual clock for each node as follows

$$VC_k(t) = PC_k(t) + \delta_k \ .$$

Here $PC_k(t)$ denotes the value read from clock PC_k at *real* time t.

To provide a global time reference, it is necessary to find (for each node k) a suitable constant, δ_k, which corrects the local clock and such that

$$| \ VC_j(t) - VC_k(t) \ | \leq \varepsilon, \ \text{ for all } j, \ k \qquad (1)$$

where ε is a sufficiently small constant. When condition (1) holds, the clocks are said to be in agreement [SRIK87]. Note that condition (1) is just Lamport's [LAMP78] condition PC2.

3.1. Synchronizing a Pair of Clocks

Consider first the problem of synchronizing a pair of processor clocks. Assume that node j has knowledge of the global time through VC_j. The goal is for node k to acquire knowledge of the global time reference using PC_k and local computations to deduce δ_k. The algorithm proceeds as follows:

[1] Personal communication with NCUBE Corp. customer service and engineering staff.

(1) Node j sends a message to node k telling node k to synchronize its clock. Node j then awaits a time request from node k.

(2) At time $t_0 = PC_k(t)$, node k sends a message to node j requesting the current global time.

(3) At time $T = VC_j(t+m)$ node j receives the request and sends the value T to node k.

(4) At time $t_1 = PC_k(t+2m)$ node k receives the response from node j, the value T.

This sequence of events is shown in Figure 2.

Now to compute δ_k note that when node j's global time was T, node k's local clock was at $t_1 - m$ where m denotes the message transit time. The difference in these two clock values is δ_k, that is,

$$\delta_k = T - (t_1 - m) \ . \tag{2}$$

Assuming that the message requesting the time from node j and the message carrying the response both took the same time m to deliver, we have $2m = (t_1 - t_0)$ which gives the following approximation for m,

$$m = \frac{t_1 - t_0}{2} \ . \tag{3}$$

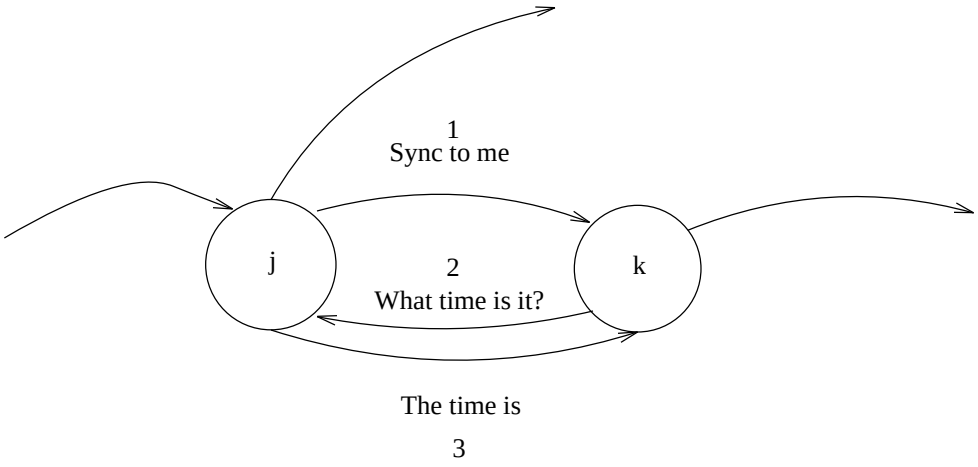

Figure 2: Clock Synchronization Message Protocol

Substituting (3) into (2) yields the desired expression for δ_k,

$$\delta_k = T - \frac{t_1 + t_0}{2} \ . \tag{4}$$

The basic assumption underlying the synchronization algorithm is that the unknown time, *m*, required to send and receive the synchronization message is constant or nearly so. This is a reasonable assumption if: (1) the length of the messages is the same; and (2) the communication channel is dedicated. Both these conditions are easily achieved on a hypercube if pairwise clock synchronizations are between adjacent nodes. We can assess the effect of this assumption as follows. Let m_0 be the transit time of the message requesting the current time and let m_1 denote the transit time of the reply. Now since

$$\delta_k \ = \ T - (t_1 - m_1) \ = \ T - (t_0 + m_0) \ ,$$

we have

$$2\delta_k = 2T - (t_1 + t_0) + (m_1 - m_0)$$

or equivalently,

$$\delta_k = T - \frac{t_1 + t_0}{2} + \frac{m_1 - m_0}{2} \ .$$

We have assumed that $m_0 = m_1 = m$. To the extent that this assumption is inaccurate, we will incur an error of $\frac{1}{2}(m_1 - m_0)$.

Since message transmission time will dominate the synchronization algorithm, it is easy to see that the execution time, T_1, of the algorithm is bounded from above by $T_1 \leq 3m$ ticks. Equality holds when the message in step 1 of the algorithm takes *m* ticks; in general, it will take less time.

3.2. Synchronizing all the Clocks

We now wish to synchronize all the processor clocks using the pairwise synchronization algorithm of the last section. The most efficient way to propagate the algorithm through all the nodes of the hypercube is to use a spanning tree. (The topological properties of hypercubes, particularly as they relate to communication, are well covered in [HO86, JOHN87, SAAD85].) This also guarantees that pairwise clock synchronizations will occur between adjacent nodes. An example of a spanning tree embedded in an order three hypercube is shown in Figure 3. The order in which the clocks of Figure 3 would be synchronized is given by the sequence

$$< (0,4), \{ (0,2), (4,6) \}, \{ (0,1), (2,3), (4,5), (6,7) \} >$$

where (i, j) signifies that node j synchronizes its clock with node i and the events in braces can occur in parallel. Notice that the algorithm terminates in $n = \log N$ [2] steps and 2^{k-1}, $k = 1, \cdots, n$ clocks are synchronized at step k.

Figure 4 illustrates a typical timing sequence resulting from running the synchronization algorithm on an order 2 cube. The vertical lines represent time as measured by each processor. For simplicity, all the clocks are shown running in phase. The directed arcs represent messages passed between the processors and correspond directly to the first three steps of the synchronization algorithm. The value in parentheses labeling the last arc of the protocol is the time T sent in

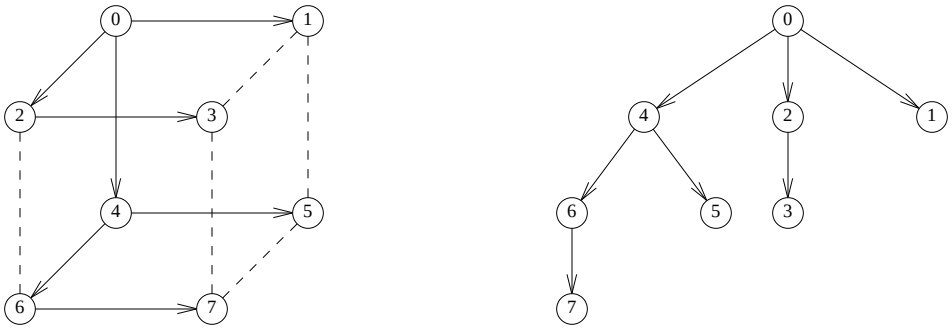

Figure 3: Spanning Tree Embedded in 3-cube

[2]Throughout, $\log x$ means $\log_2 x$.

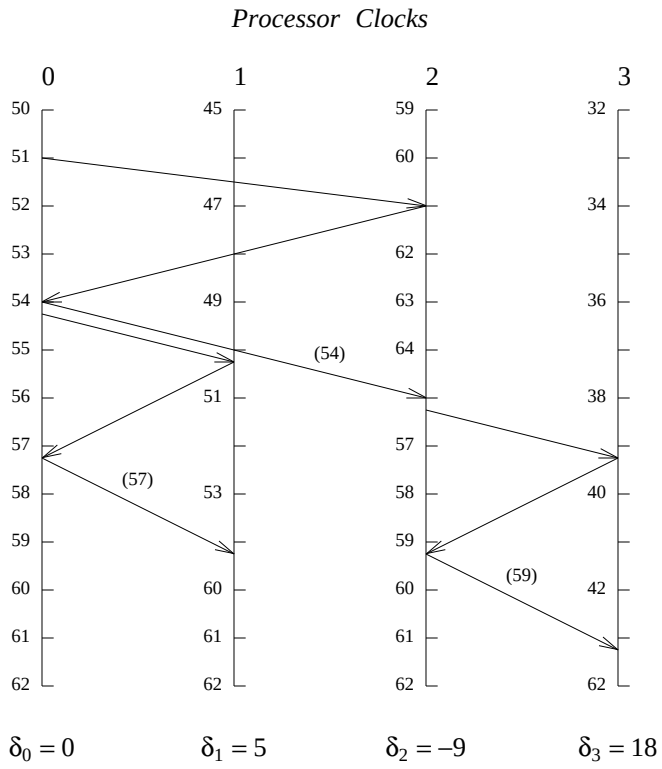

Figure 4: 2-cube Clock Synchronization Example

step 3 of the algorithm. The small time increment between the termination of one synchronization and the initiation of the next is meant to suggest a small computation (e.g., the calculation of δ_k) occurring between messages. The order in which the clock synchronizations occur is given by

$$< (0,2), \{ (0,1), (2,3) \} >$$

and the calculated values for the δ_k are shown at the bottom of the figure. The values shown can easily be verified from the diagram, for example,

$$\delta_1 = 57 - \frac{54 + 50}{2} = 5 \ .$$

Let T_n denote the time required to execute the n stages of the synchronization algorithm. Since $T_1 \leq 3m$, it follows that $T_n \leq 3m \log N = 3mn$. In the example of Fig. 4, $T_n = 11 \leq 3mn = 12$. Notice that although the time line for node 0 in Figure 4 suggests that the nodes are capable of

simultaneous message transmission along multiple channels, this is not a requirement of the algorithm. The algorithm executes in the same time even when messages on the same processor cannot be overlapped.

3.3. Establishing a Base for the Time Reference

VC_0, the clock on node 0 [3], is the clock to which all other clocks are ultimately synchronized. Therefore, the value of VC_0 determines the base of the global time reference. Since $VC_0 = PC_0 + \delta_0$, we can easily determine the base of the time reference by appropriate choice of the initial value for δ_0. Several possibilities exist. If we wish to bring the remaining clocks into synchrony with the current local time on node 0, we would choose $\delta_0 = 0$. If, however, we wanted to start a new epoch at time 0, we would set $\delta_0 = -PC_0(t)$.

Another possibility results from the desire to resynchronize the system of clocks. If for some reason the clocks have drifted away from an acceptable state, we could reset them to some time in the virtual future. One way to do this is to find $\max_k VC_k$ and assign the role of node 0 to that node. The synchronization algorithm is then run with $\delta_0 = VC_0 + 3mn$. Although this creates a discontinuity in the virtual time sequence, it safely avoids unintentionally reproducing the virtual past on any node.

One final note. Since the node clocks are implemented by fixed length integer counters, they have a period determined by their update rate. This will typically be in the 5 to 50 day range. The global time reference may have a period substantially shorter than the underlying clock depending on the choice of δ_0 and the current value of the system clock. This is not a serious problem and can be easily worked around.

[3] Note that the node chosen to be node 0 is quite arbitrary; it is the role of node 0 which is important not the particular processor.

3.4. Clock Agreement

When discussing clock agreement, we should distinguish between (a) the agreement attainable immediately after running a synchronization algorithm, and (b) the instantaneous agreement of the clock ensemble, that is, the observable agreement of the clocks after the passage of an arbitrary interval of time. The initial agreement of the clocks will be due to the synchronization algorithm while the ability to maintain this agreement will depend on the drift characteristics of the clocks.

Lamport [LAMP78] has characterized a discrete clock as "a continuous one in which there is an error of up to ½ tick in reading it." Thus, when reading a discrete clock we have an error bounded by

$$0 \leq t - VC_k(t) < 1 \ .$$

This means that the best agreement we can expect is

$$| \ VC_j(t) - VC_k(t) \ | \leq 1 \ . \tag{5}$$

Since the longest path in the spanning tree is $\log N$, we would expect the worst case clock agreement to be

$$| \ VC_j(t) - VC_k(t) \ | \leq \log N, \ \text{ for all } j, \ k \ .$$

This follows from (5) where we consider that an error of one tick is introduced at each synchronization step. However, since the probability that a one tick error will be introduced is independent of the accumulated error, the expected agreement of the clocks should be

$$| \ VC_j(t) - VC_k(t) \ | \approx \frac{\log N}{2}, \ \text{ for all } j, \ k \ . \tag{6}$$

Empirical evidence is given in the next section to support this conjecture. A more formal justification of (6) and a general analysis of error propagation will be the subject of a later paper.

4. Accuracy and Stability - Empirical Results

We now turn our attention to the evaluation of the quality of the global time reference achieved by using the algorithm of the last section. Two properties of interest are the *stability* and *accuracy* of the clock ensemble. By stability we mean the relative immunity of the separate clocks to drift relative to one another. Clearly this is a property of the hardware timing mechanism and is independent of the synchronization algorithm. The accuracy of the global time reference is an evaluation of the pairwise agreement (cf. inequality 1) of the individual clocks. In order to assess these properties, we must sample the individual processor clocks.

4.1. Sampling the Processor Clocks

After we have run the clock synchronization protocol, it is no longer necessary to preserve the distinction between the physical and virtual clocks. We simply assume a single clock, C_k, on each node k and denote the reading of the clock at time t by $c_k(t)$. Ideally, we would like to read all the processor clocks at exactly the same instant in time, that is, at time t we read all the $c_k(t)$. Because we have to ask each processor to report its time and messages incur delays, this clearly cannot be done on a message passing multicomputer. Accordingly, each clock must be sampled individually and message latencies must be accounted for in the sampling procedure. A ring topology was chosen for this sampling for two reasons. First, to minimize experimental bias a topology different from that used to propagate the synchronization protocol is desirable. Secondly, a ring topology is easy to analyze and does not introduce timing side effects (such as overlapped message transmission) which might occur using other interconnection schemes.

A ring topology is easy to embed in a hypercube by using a gray code to construct the ring. Figure 5 illustrates such an embedding arising from a simple binary reflected gray code. For simplicity of notation, we will regard node k ($k = 0, 1, \cdots, n-1$) in the ring topology to be the k^{th} node in the ring, that is, at distance k from node 0, and ignore the actual processor labels shown in Figure 5. For example, node 0 is processor 0, node 3 is processor 2, node 5 is processor 7, etc.

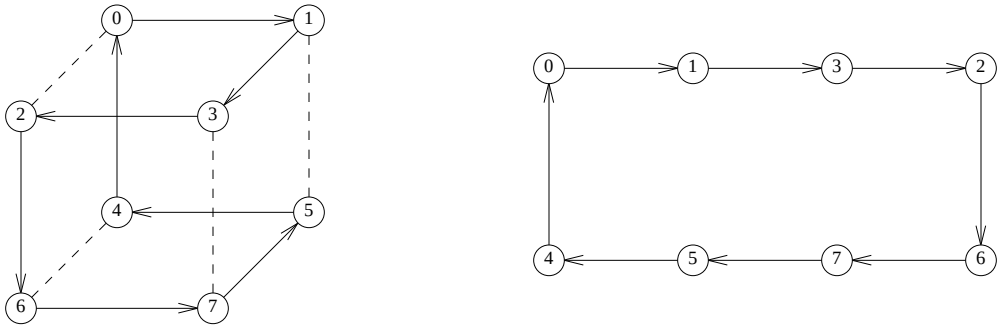

Figure 5: Ring Topology Embedded in 3-cube

The sampling method consists of first sending a message from node 0 around the ring requesting that each processor record the message arrival time. In the second step, a request for the sampled values is broadcast to all nodes and finally, in the third step, all the values are sent to node 0 for analysis.

The outcome of one such trial on an order 3 cube is shown in Table 1. The values shown in Table 1(a) were sampled immediately after the synchronization protocol was run. The values in Table 1(b) were sampled 12 hours later. (Note that the actual values shown in Table 1(b) have been reduced by 307576093 ticks for easier visual comparison.) The actual clock values sampled, denoted by $c_k(t+km)$, are shown in the second column of the table. The remaining entries in the table are described below.

For any node k, we have

$$c_k(t+km) = c_k(t) + km \quad . \tag{7}$$

Since the internode message transit time, m, is unknown, we use the following estimate

$$\overline{m} = \frac{c_0(t+Nm) - c_0(t)}{N} \tag{8}$$

where $c_0(t+Nm)$ is the time the sampling message returns to node 0. One is tempted to estimate

k	$c_k(t+km)$	$\hat{c}_k(t+km)$	$\lfloor +.5 \rfloor$	$\hat{c}_k(t)$	$\lceil -.5 \rceil$
0	146	146	146	146	146
1	154	154.125	154	145.875	146
2	163	162.25	162	146.75	147
3	171	170.375	170	146.625	147
4	179	178.5	179	146.5	146
5	188	186.625	187	147.375	147
6	195	194.75	195	146.25	146
7	203	202.875	203	146.125	146

$\overline{m} = 8.125$, range = 1.5

(a) Sample Immediately after Synchronization

k	$c_k(t+km)$	$\hat{c}_k(t+km)$	$\lfloor +.5 \rfloor$	$\hat{c}_k(t)$	$\lceil -.5 \rceil$
0	146	146	146	146	146
1	155	154.25	154	146.75	147
2	163	162.5	163	146.5	146
3	171	170.75	171	146.25	146
4	180	179	179	147	147
5	188	187.25	187	146.75	147
6	196	195.5	196	146.5	146
7	204	203.75	204	146.25	146

$\overline{m} = 8.25$, range = 1.0

(b) Sample 12 Hours Later

Table 1: Clock Stability and Accuracy

m locally on each node as

$$\overline{m}_k = \frac{c_k(t+km) - c_0(t)}{k} \quad .$$

But notice that, together with eqn. (7), this guarantees that $c_k(t) = c_0(t)$ by construction. This emphasizes the importance of measuring intervals with respect to the same clock.

Now if the clocks are truly synchronized, we have by hypothesis $c_k(t) = c_0(t)$ for all k. Hence, we can estimate the message arrival time on any node k by

$$\hat{c}_k(t+km) = c_0(t) + k\overline{m} \quad .$$

This expression estimates the time at which the sampling message arrived at node k relative to the clock on node 0. Similarly, we can estimate the time the sampling message left node 0 relative to node k's clock by

$$\hat{c}_k(t) = c_k(t+km) - k\bar{m} \ .$$

We obtain integer estimates of the clock readings by rounding the estimates of $\hat{c}_k(t+km)$ and $\hat{c}_k(t)$. This rounding is motivated by the fact that although we regard m as constant, it need not be integral. The rounding strategy used is as follows:

$$\hat{c}_k(t+km) = \left\lfloor c_0(t) + k\bar{m} + .5 \right\rfloor \quad \text{and} \quad \hat{c}_k(t) = \left\lceil c_k(t+km) - k\bar{m} - .5 \right\rceil \ .$$

The only difference in the rounding method for these quantities is that .5 is rounded up in one direction and down in the other.

Table 1 shows the calculated values of these clock estimates together with the observed values of $c_k(t+km)$. A glance at the last column of Table 1(a) and 1(b) should convince the reader that the clock ensemble was maintaining an extremely accurate global time reference for the 12 hour time period monitored. This behavior was consistently observed for all cube orders tested.

As we mentioned in the beginning of this section, stability is a hardware property. Our investigation showed that the NCUBE/ten provides a very stable timing source and we could find no statistically significant difference in the behavior of the clock ensemble when sampling the clocks every 10 minutes for 12 hours and when sampling every 10 seconds for 12 minutes.

To test the agreement of the clock ensemble it is necessary to calculate the sample range of the $c_k(t)$, which is given by

$$\max_k \hat{c}_k(t) - \min_k \hat{c}_k(t) \ .$$

This has been done for the data of Table 1 and is shown in the table.

In order to get a feel for the expected agreement of the clock ensemble the following experiment was conducted. For each order n hypercube ($1 \le n \le 6$), a program was run to synchronize the clocks and then sample the clocks every 10 seconds for 12 minutes. For each clock sample, the sample range was calculated. This procedure was repeated 10 times resulting in 720 observations from which a pooled estimate of the expected value of the range was calculated.

The results of these calculations are shown in the plot of Figure 6. The line $n/2$ is shown as a dashed line in the figure for comparison. The data give strong empirical evidence to the conjecture (see (6) in Section 3) that the expected agreement of the clocks is $\frac{\log N}{2} = \frac{n}{2}$ ticks.

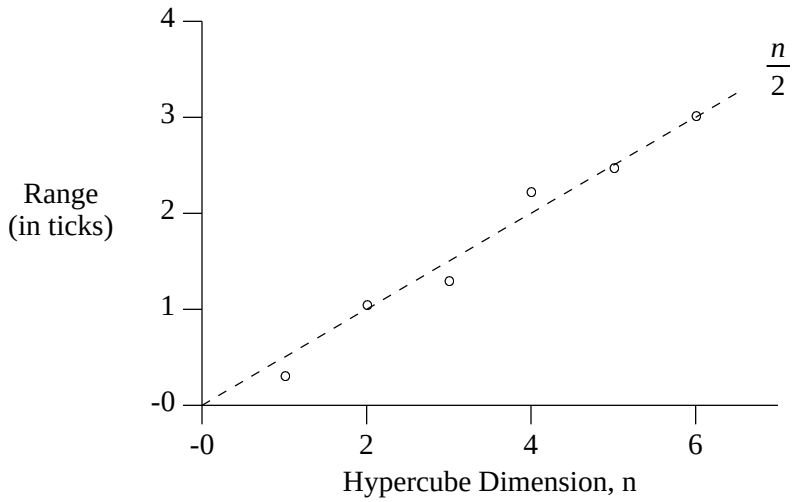

Figure 6: Expected Clock Agreement

4.2. One Pass Sampling Technique

The detailed sampling of the last section is far too expensive to use as the basis of a decision procedure for ascertaining whether or not the agreement of a system of clocks is "good enough." The sampling method used involves passing at least $3N-2$ messages and takes at least $N+2\log N$ steps to complete.[4] This would be acceptable if the need to sample was infrequent. However, in systems with more clock drift, the need to sample frequently demands an efficient procedure in order to be viable. In this section, we describe a method for computing the mean, $\overline{c}(t)$, and variance, s^2, of the $c_k(t)$, which passes N messages in N steps around a ring.

The idea is to collect enough summary information in one pass around a ring to be able to deduce $\overline{c}(t)$ and s^2 when the message returns to node 0. Clearly this is trivial if the internode message transit time, m, is known. Here we treat the case where m is constant but unknown. The basis of the method is as follows. First, let $\Delta_k = c_k(t+km) - c_0(t)$ and define the statistics S_1, S_2, and S_3 as

$$S_1 = \sum_{k=0}^{N-1} \Delta_k , \quad S_2 = \sum_{k=0}^{N-1} k\Delta_k , \quad \text{and} \quad S_3 = \sum_{k=0}^{N-1} {\Delta_k}^2 .$$

Given knowledge of these sums, it can be shown that

$$\overline{c}(t) = c_0(t) + \frac{S_1}{N} - \frac{(N-1)}{2}m$$

and

$$s^2 = \frac{NS_3 - {S_1}^2}{N(N-1)} + m\left[S_1 - \frac{2S_2}{N-1} + \frac{mN(N+1)}{12}\right] .$$

The derivation of these quantities is straightforward, but tedious. The details can be found in Appendix A.

---

[4] It takes N messages (N steps) to propagate the sampling packet around the ring. After the packet arrives back at node 0, $N-1$ messages are sent ($\log N$ steps) to request the sampled values. Finally, $N-1$ messages ($\log N$ steps) are required to report the sampled values to node 0.

The method is now simply: initiate a message from node 0 containing the value $c_0(t)$ and initial values of 0 for the sums S_1, S_2, and S_3; as the message arrives at each node *k*, the clock is sampled and the values of the partial sums updated. When the message arrives back at node 0, the sums can be used, together with the estimate of $\overline{m}$ given in equation (8), to compute the sample mean and variance. An example of these calculations is shown in Table 2. The entries in Table 2 derive from the data in Table 1 and can be verified by independent calculation using that data.

Some justification should be given for the choice of Δ_k. This quantity was defined simply to slow the growth of the sum S_3. It is easy to see that S_3 is $O(m^2N^3)$ and a quick calculation shows that for reasonable values of *m*, this algorithm can be used on systems of up to $N=512$ processors using unsigned 32 bit arithmetic. On systems with more than 512 processors we would have to use floating point arithmetic which would increase the size of the message, the length of the computation, and hence, increase the message latency.

We have experimented with some decision procedures based on the sample mean and variance defined above. In addition, we are looking at simpler procedures which exploit particular properties (see Appendix B) of the underlying distribution. The basic idea is to bound the sample range so that a particular level of clock agreement is achieved with some confidence. We anticipate that results on distribution-free bounds of order statistics [DAVI81] can be used to refine the decision procedure. This work is continuing and will be covered in a subsequent report.

$\overline{m}$	S_1	S_2	S_3	$\overline{c}(t)$	s
8.125	231	1152	9481	146.4375	.486
8.25	235	1170	9779	146.5	.327

Table 2: Sample Calculations from Table 1 Data

4.3. NCUBE/AT Timing Anomalies

As a practical example of the utility of a global clock as a diagnostic tool, this section describes some timing anomalies associated with the NCUBE/AT hypercube which were uncovered with the aid of the global clock service described in this paper.

Two test programs were run on the NCUBE/AT to gather clock samples under slightly different conditions. These programs sampled the global clock five times at 10 second intervals. The clock samples obtained at each sampling interval were normalized by subtracting the sample minimum from all the sample values. The results of these tests from two different NCUBE/AT order 2 hypercubes are summarized in Table 3 and described below. The rows in the table correspond to one sample of all the node clocks; the columns show the times recorded for a single node. Note that the ideal case (i.e., a perfectly synchronized system of clocks) would yield zero for all the normalized clock values.

The first test program (A) is simply the sampling program described earlier. Recall that the sampling method consists of first sending a message around the ring requesting that each proces-

NCUBE/AT Timing Anomalies								
Test	Machine 1				Machine 2			
	0	1	2	3	0	1	2	3
A	0	1	1	0	0	0	0	1
	0	3	3	3	0	3	3	3
	0	6	5	6	0	5	5	6
	0	9	8	9	0	7	9	8
	0	11	11	11	0	9	11	11
B	0	0	0	1	0	0	1	1
	43	0	0	0	43	0	0	1
	88	1	0	1	87	1	1	0
	132	0	1	0	131	1	1	0
	177	1	2	0	176	1	1	0

Table 3: NCUBE/AT Timing Anomalies

sor record the message arrival time. A request for the sampled values is then broadcast to all nodes and finally, in the last step, all the sampled values are sent to node 0 for analysis.

From the table we see that: (1) node 0 consistently has the minimum clock value; and (2) the remaining three nodes are synchronized with respect to one another, but are apparently advancing (running faster than node 0) at a rate of approximately 2 ticks per sample. The only difference between the processing on node 0 and the remaining nodes is in the last step where node 0 is the recipient of all the sampled values. Apparently this burst of message activity is causing node 0 to fail to update its clock appropriately. One possible explanation for this phenomenon is that node 0 may be operating with its clock interrupt disabled for all or some portion of the message receiving process.

The second test program (B) is just a modification of program A. Program A was modified so that in addition to the behavior described above, each node sends its sampled value to the display screen. From the table we see that: (1) node 0 is apparently running faster than the remaining nodes at a rate of approximately 44 ticks per sample; and (2) the remaining three nodes are synchronized. Apparently the clocks on nodes 2-3 are disabled for 44 ticks during each sample period. This discrepancy can only be attributed to the screen I/O since test programs A and B differ only in this respect.

Since the node clocks seem to be affected by (1) message loads and (2) interactions with AXIS, the conclusion to be drawn from the data of Table 3 is inescapable: in general, no timing information can be reliably obtained from the NCUBE/AT. This means that simple interval timing can not be done accurately unless there is no message traffic.

Recall that the NCUBE/AT and the NCUBE/ten use the same node processor chips. Also, the NCUBE/ten communication is done via VORTEX and VERTEX, while the NCUBE/AT uses a combination node OS called VOERTEX. Since the NCUBE/ten does not exhibit the same timing anomalies, it seems reasonable to infer that VOERTEX is responsible for the problem.

5. Conclusions and Future Work

The simple clock synchronization algorithm described in this paper was shown to provide a very high quality global time reference on the NCUBE/ten hypercube. The stability of the clock is due to the single timing source provided on the NCUBE. The excellent agreement of the clock ensemble is due to the synchronization algorithm. Empirical evidence was given to support the conjecture that the expected agreement of the clock ensemble after synchronization is $\frac{\log N}{2} = \frac{n}{2}$ ticks.

The algorithm is efficient enough to be used as part of the normal node program load sequence to provide application programs with a global clock initialized to time 0. At the very least, it could be used during the node bootstrap process (*ninit* on the NCUBE hardware) to establish a single system wide time base.

The utility of a global time reference as a diagnostic tool was also demonstrated when timing anomalies in the NCUBE/AT were uncovered. We speculate that the NCUBE/AT node OS, VOERTEX, is causing the problems.

We intend to duplicate the experiments described in this report on the Intel iPSC/2 hypercube as well as on an order-2 cube made up of PC's cabled together through serial ports. Each node of the Intel hypercube has its own crystal and therefore, the clock ensemble should exhibit much more drift than the NCUBE hardware. In addition to separately clocked nodes, the PC system will offer more variability in message transmission time. These new environments will provide valuable insight into the clock synchronization problem. They will enable us to access the effects of more loosely coupled systems with greater potential for clock drift and variability in message latency, on the maintenance of a global time service.

Finally, the techniques described in this paper should be applicable to multicomputers using different communication topologies. If the graph of the communication network has diameter D, the expected agreement of the clocks after synchronization should be $\frac{1}{2}D\varepsilon$ where ε is an upper

bound on the pairwise agreement of the individual clocks. In the case of the hypercube, $D = n$ and we have given empirical evidence to support the claim that $\varepsilon = 1$.

APPENDIX A

When sampling the clocks in a hypercube multicomputer using a ring topology, each node k will record the observation $c_k(t+km)$, which is the time of arrival of the sampling message with respect to node k's clock. Now $c_k(t+km) = c_k(t) + km$, where $c_k(t)$ is the time the message left node 0 with respect to node k's clock and m is the internode message transit time. We want to evaluate the sample mean, $\overline{c}(t)$, and the sample variance, s^2, of the $c_k(t)$. In this appendix we show how to derive $\overline{c}(t)$ and s^2 from information gathered in one pass around the ring when the message transit time, m, is unknown.

Let $\Delta_k = c_k(t+km) - c_0(t)$ and consider the statistics

$$S_1 = \sum_{k=0}^{N-1} \Delta_k , \quad S_2 = \sum_{k=0}^{N-1} k\Delta_k , \quad \text{and} \quad S_3 = \sum_{k=0}^{N-1} {\Delta_k}^2 .$$

If $c_k(t) = c_0(t)$, then Δ_k can be interpreted as the elapsed time of the message since it left node 0. Alternatively, Δ_k can be interpreted as a scaling operation designed to keep the sums S_1, S_2, and S_3 from growing too rapidly.

The sample mean can be derived as follows:

$$\begin{aligned} S_1 &= \sum_{k=0}^{N-1} \Delta_k \\ &= \sum_{k=0}^{N-1} \left(c_k(t+km) - c_0(t) \right) \\ &= \sum_{k=0}^{N-1} \left(c_k(t) + km \right) - Nc_0(t) \\ &= \sum_{k=0}^{N-1} c_k(t) + m \sum_{k=0}^{N-1} k - Nc_0(t) \\ &= \sum_{k=0}^{N-1} c_k(t) + \frac{N(N-1)}{2} m - Nc_0(t) = N\left(\overline{c}(t) + \frac{(N-1)}{2} m - c_0(t) \right) \end{aligned} \tag{A-1}$$

from which it follows that

$$\overline{c}(t) = c_0(t) + \frac{S_1}{N} - \frac{(N-1)}{2}m \ . \tag{A-2}$$

We now derive the sample variance. We begin by expanding the sum S_2. This sum is used later to simplify S_3.

$$\begin{aligned}
S_2 &= \sum_{k=0}^{N-1} k\Delta_k \\
&= \sum_{k=0}^{N-1} k\Big[c_k(t+km) - c_0(t) \Big] \\
&= \sum_{k=0}^{N-1} k\Big[c_k(t) + km - c_0(t) \Big] \\
&= \sum_{k=0}^{N-1} kc_k(t) + m\sum_{k=0}^{N-1} k^2 - c_0(t)\sum_{k=0}^{N-1} k
\end{aligned} \tag{A-3}$$

Turning now to S_3 we have

$$\begin{aligned}
S_3 &= \sum_{k=0}^{N-1} {\Delta_k}^2 \\
&= \sum_{k=0}^{N-1} \Big[c_k(t+km) - c_0(t) \Big]^2 \\
&= \sum_{k=0}^{N-1} \Big[c_k(t+km)^2 - 2c_k(t+km)c_0(t) + c_0(t)^2 \Big] \\
&= \sum_{k=0}^{N-1} \Big[[c_k(t) + km]^2 - 2[c_k(t) + km]c_0(t) + c_0(t)^2 \Big] \\
&= \sum_{k=0}^{N-1} \Big[c_k(t)^2 + 2kmc_k(t) + (km)^2 - 2c_0(t)c_k(t) - 2kmc_0(t) + c_0(t)^2 \Big] \\
&= \sum_{k=0}^{N-1} c_k(t)^2 + 2m\sum_{k=0}^{N-1} kc_k(t) + m^2\sum_{k=0}^{N-1} k^2 - 2c_0(t)\sum_{k=0}^{N-1} c_k(t) - 2mc_0(t)\sum_{k=0}^{N-1} k + Nc_0(t)^2 \\
&= \sum_{k=0}^{N-1} c_k(t)^2 + 2m\Big[\sum_{k=0}^{N-1} kc_k(t) + m\sum_{k=0}^{N-1} k^2 - c_0(t)\sum_{k=0}^{N-1} k \Big] - m^2\sum_{k=0}^{N-1} k^2 - 2c_0(t)\sum_{k=0}^{N-1} c_k(t) + Nc_0(t)^2
\end{aligned}$$

Substituting (A-3) gives

$$S_3 = \sum_{k=0}^{N-1} c_k(t)^2 + 2mS_2 - m^2 \sum_{k=0}^{N-1} k^2 - 2c_0(t) \sum_{k=0}^{N-1} c_k(t) + Nc_0(t)^2$$

We can now remove the $\sum_{k=0}^{N-1} c_k(t)$ term using (A-1) to get

$$S_3 = \sum_{k=0}^{N-1} c_k(t)^2 + 2mS_2 - m^2 \sum_{k=0}^{N-1} k^2 - 2c_0(t) \left\{ S_1 - m \sum_{k=0}^{N-1} k + Nc_0(t) \right\} + Nc_0(t)^2$$

$$= \sum_{k=0}^{N-1} c_k(t)^2 + 2mS_2 - m^2 \sum_{k=0}^{N-1} k^2 - 2c_0(t) \left\{ S_1 - m \sum_{k=0}^{N-1} k \right\} - Nc_0(t)^2 \qquad \text{(A-4)}$$

The sample variance, s^2, is given by

$$s^2 = \frac{N \sum_{k=0}^{N-1} c_k(t)^2 - \left\{ \sum_{k=0}^{N-1} c_k(t) \right\}^2}{N(N-1)} \quad .$$

Multiplying (A-4) by N and rearranging gives

$$N \sum_{k=0}^{N-1} c_k(t)^2 = NS_3 - 2mNS_2 + m^2 N \sum_{k=0}^{N-1} k^2 + 2Nc_0(t) \left\{ S_1 - m \sum_{k=0}^{N-1} k \right\} + N^2 c_0(t)^2 \qquad \text{(A-5)}$$

From (A-1) we have

$$\left\{ \sum_{k=0}^{N-1} c_k(t) \right\}^2 = \left\{ S_1 - m \sum_{k=0}^{N-1} k + Nc_0(t) \right\}^2$$

$$= {S_1}^2 - 2mS_1 \sum_{k=0}^{N-1} k + \left\{ m \sum_{k=0}^{N-1} k \right\}^2 + 2Nc_0(t) \left\{ S_1 - m \sum_{k=0}^{N-1} k \right\} + N^2 c_0(t)^2 \qquad \text{(A-6)}$$

Subtracting (A-6) from (A-5) yields

$$N(N-1)s^2 = N \sum_{k=0}^{N-1} c_k(t)^2 - \left\{ \sum_{k=0}^{N-1} c_k(t) \right\}^2$$

$$= NS_3 - {S_1}^2 - 2mNS_2 + 2mS_1 \sum_{k=0}^{N-1} k + m^2 N \sum_{k=0}^{N-1} k^2 - \left\{ m \sum_{k=0}^{N-1} k \right\}^2$$

$$= NS_3 - {S_1}^2 - 2m \left\{ NS_2 - S_1 \sum_{k=0}^{N-1} k \right\} + m^2 \left\{ N \sum_{k=0}^{N-1} k^2 - \left\{ \sum_{k=0}^{N-1} k \right\}^2 \right\} \qquad \text{(A-7)}$$

Equation (A-7) can be simplified using the identities

$$\sum_{k=1}^{N} k = \frac{N(N+1)}{2} \quad \text{and} \quad \sum_{k=1}^{N} k^2 = \frac{N(N+1)(2N+1)}{6} \ ,$$

so that

$$N\sum_{k=0}^{N-1} k^2 - \left[\sum_{k=0}^{N-1} k\right]^2 = \frac{N^2(N-1)(2N-1)}{6} - \left[\frac{N(N-1)}{2}\right]^2$$

$$= N^2(N-1)\left[\frac{2N-1}{6} - \frac{N-1}{4}\right]$$

$$= \frac{N^2(N-1)(N+1)}{12}$$

and

$$S_1 \sum_{k=0}^{N-1} k = \frac{N(N-1)}{2} S_1 \ .$$

Now equation (A-7) can be written as

$$N(N-1)s^2 = NS_3 - {S_1}^2 - 2m\left[NS_2 - \frac{N(N-1)}{2} S_1\right] + \frac{m^2N^2(N-1)(N+1)}{12}$$

$$= NS_3 - {S_1}^2 + mN(N-1)S_1 - 2mNS_2 + \frac{m^2N^2(N-1)(N+1)}{12}$$

from which it follows that

$$s^2 = \frac{NS_3 - {S_1}^2}{N(N-1)} + m\left[S_1 - \frac{2S_2}{N-1} + \frac{mN(N+1)}{12}\right] \ . \qquad \text{(A-8)}$$

In both (A-2) and (A-8) we use the following estimate for the unknown internode message transit time, m,

$$\overline{m} = \frac{c_0(t+Nm) - c_0(t)}{N} \ .$$

APPENDIX B

In this appendix, we develop the relationship between the observed clock values and the estimated values given by

$$\hat{c}_k(t+km) = c_0(t) + km \ , \tag{B-1}$$

the estimated clock value on node *k* relative to node 0's clock, and

$$\hat{c}_k(t) = c_k(t+km) - km \ , \tag{B-2}$$

the estimated value of node *k*'s clock at time *t*. Consider the sum of the differences between each clock's observed and estimated value.

$$\begin{aligned} \sum_{k=0}^{N-1} \left[c_k(t+km) - \hat{c}_k(t+km) \right] &= \sum_{k=0}^{N-1} \left[c_k(t+km) - c_0(t) \right] - m \sum_{k=0}^{N-1} k \\ &= S_1 - m \sum_{k=0}^{N-1} k \end{aligned} \tag{B-3}$$

where S_1 is just the sum defined in Appendix A. Comparing (B-3) with (A-1) we find that

$$\begin{aligned} \sum_{k=0}^{N-1} \left[c_k(t+km) - \hat{c}_k(t+km) \right] &= \sum_{k=0}^{N-1} c_k(t) - N c_0(t) \\ &= N[\bar{c}(t) - c_0(t)] \end{aligned} \tag{B-4}$$

From (B-1) and (B-2), we also have that

$$c_k(t+km) - \hat{c}_k(t+km) \ = \ c_k(t+km) - [c_0(t) + km \ ] \ = \ \hat{c}_k(t) - c_0(t)$$

which, together with (B-4), implies that

$$\sum_{k=0}^{N-1} \left[\hat{c}_k(t) - c_0(t) \right] = N[\bar{c}(t) - c_0(t)] \tag{B-5}$$

This expression is the sum of the individual clock deviations from $c_0(t)$.

From (B-4) and (B-5) we see that the expression

$$\bar{c}(t) - c_0(t) \tag{B-6}$$

can be interpreted as: (1) the average deviation between the observed and estimated values of any particular clock; or (2) the average deviation of any clock from $c_0(t)$. More generally, we can regard (B-6) as the average deviation of the entire clock ensemble from the hypothetically true clock time $c_0(t)$.

References:

[DAVI81] H. A. David, *Order Statistics*, John Wiley & Sons, Inc., New York, NY, second edition 1981.

[HAYE86a] J. P. Hayes, T. N. Mudge, Q. F. Stout, S. Colley and J. Palmer, ‘‘Architecture of a Hypercube Supercomputer’’, *Proc. of the International Conference on Parallel Processing*, Aug. 1986, 653-660.

[HAYE86b] J. P. Hayes, T. N. Mudge, Q. F. Stout, S. Colley and J. Palmer, ‘‘A Microprocessor-based Hypercube Supercomputer’’, *IEEE Micro*, Oct. 1986, 6-17.

[HO86] C. T. Ho and S. L. Johnsson, ‘‘Distributed Routing Algorithms for Broadcasting and Personalized Communication in Hypercubes’’, *Proc. of the International Conference on Parallel Processing*, Aug. 1986, 640-648.

[JOHN87] S. L. Johnsson and C. T. Ho, ‘‘Optimum Broadcasting and Personalized Communication in Hypercubes’’, Tech. Rep. YALEU/DCS/RR-610, Dept. of Computer Science, Yale University, Dec. 1987.

[JONE80] A. K. Jones and P. Schwarz, ‘‘Experience Using Multiprocessor Systems - A Status Report’’, *Computing Surveys 12*, 2 (June 1980), 121-164.

[LAMP78] L. Lamport, ‘‘Time, Clocks, and the Ordering of Events in a Distributed System’’, *Comm. ACM 21*, 7 (July 1978), 558-565.

[LAMP84] L. Lamport, ‘‘Using Time Instead of Timeout for Fault-Tolerant Distributed Systems’’, *ACM Trans. Prog. Lang. and Systems 6*, 2 (Apr. 1984), 254-280.

[LAMP85] L. Lamport and P. M. Melliar-Smith, ‘‘Synchronizing Clocks in the Presence of Faults’’, *J. ACM 32*, 1 (Jan. 1985), 52-78.

[SAAD85] Y. Saad and M. H. Schultz, ‘‘Data Communication in Hypercubes’’, Tech. Rep. YALEU/DCS/RR-428, Dept. of Computer Science, Yale University, Oct. 1985.

[SAAD88] Y. Saad and M. H. Schultz, ‘‘Topological Properties of Hypercubes’’, *IEEE Trans. on Computers 37*, 7 (July 1988), 867-872.

[SHIN87] K. G. Shin and P. Ramanathan, ‘‘Clock Synchronization of a Large Multiprocessor System in the Presence of Malicious Faults’’, *IEEE Trans. on Computers C-36*, 1 (Jan. 1987), 2-12.

[SRIK87] T. K. Srikanth and S. Toueg, ‘‘Optimal Clock Synchronization’’, *J. ACM 34*, 3 (July 1987), 626-645.

[TANE85] A. S. Tanenbaum and R. van Renesse, ‘‘Distributed Operating Systems’’, *Computing Surveys 17*, 4 (Dec. 1985), 419-470.

[VASA88] N. Vasanthavada and P. N. Marinos, ‘‘Synchronization of Fault-Tolerant Clocks in the Presence of Malicious Failures’’, *IEEE Trans. on Computers 37*, 4 (Apr. 1988), 440-448.

Table of Contents