Fault-Tolerant Clock Synchronization Validation Methodology

Ricky W. Butler,* Daniel L. Palumbo,* and Sally C. Johnson*
NASA Langley Research Center, Hampton, Virginia

A validation method for the synchronization subsystem of a fault-tolerant computer system is presented. The high reliability requirement of flight-crucial systems precludes the use of most traditional validation methods. The method presented utilizes formal design proof to uncover design and coding errors and experimentation to validate the assumptions of the design proof. The experimental method is described and illustrated by validating the clock synchronization system of the Software Implemented Fault Tolerance computer. The design proof of the algorithm includes a theorem that defines the maximum skew between any two nonfaulty clocks in the system in terms of specific system parameters. Most of these parameters are deterministic. One crucial parameter is the upper bound on the clock read error, which is stochastic. The probability that this upper bound is exceeded is calculated from data obtained by the measurement of system parameters. This probability is then included in a detailed reliability analysis of the system.

Nomenclature

$a_{qp}(T)$ = actual skew between clocks p and q in real time at clock time T
$A_{qp}(t)$ = difference between clocks p and q at real time t
B = maximum hardware broadcast time
C = parameter of the Zipf distribution
$\hat{C}$ = maximum likelihood estimator for C
$C(t)$ = function mapping real time into clock time
$C_p^{(i)}$ = processor p's clock function during i^{th} interval
e_{qp} = processor p's error in reading processor q's clock
$E(\ )$ = expected value of a random variable
f = number of processors with faults not affecting their clocks
m = number of faulty clocks
N = number of clocks in system
p_i = probability of assumption i being violated
p_ε = probability of read error $> \varepsilon$ for a single clock read
Prob[ ] = statistical probability of a specified event occurring
$r(T)$ = function mapping real time into clock time
r_p = processor p's clock function
R = synchronization interval
$R^{(i)}$ = i^{th} synchronization interval
S = execution time for synchronization task
$S^{(i)}$ = last S seconds of i^{th} synchronization interval
t = real time
T = clock time
v = communication delay estimate
W = length of clock read window
X_{qp} = exact delay for communication from processor q to processor p
α = parameter of Zipf distribution
$\hat{\alpha}$ = maximum likelihood estimator for α
γ = fraction of hardware faults that inhibit correct clock function
δ = maximum clock skew
δ_0 = initial clock skew at $T^{(0)}$
Δ_p = clock correction for processor p
Δ_{qp} = processor p's perception of processor q's skew with respect to itself
$\Delta_p^{(i)}$ = i^{th} clock correction for processor p
ε = bound on clock read error
ε_v = constraint on ε caused by prevote delay
ε_Ω = constraint on ε caused by maximum acceptable skew
ε_w = constraint on ε caused by clock read window
ϕ_c = reconfiguration rate for fault affecting clock
ϕ_p = reconfiguration rate for processor fault not affecting clock
η = number of reads performed within time T
λ_i = failure rate of assumption i
λ_p = failure rate of processors
μ = bias due to error in choosing v
ψ = prevote delay
ρ = bound on drift rate of nonfaulty clock
Ω = maximum acceptable skew between clocks

Received Sept. 4, 1986; revision received Feb. 2, 1987.

*Aerospace Technologist, System Validation Methods Branch, Information Systems Division.

Introduction

HIGHLY reliable flight control systems require an exact consensus to perform fault isolation and reconfiguration in real time. Inexact voting is unsuitable because of its low fault coverage and high false alarm rate. Many reconfigurable computer systems utilize an exact-match voting algorithm that depends critically upon the synchronization of the redundant computing elements.[1] In such systems, the entire communication mechanism depends fundamentally on maintaining adequate synchronization between the system clocks. Therefore, any validation of a fault-tolerant computer system must include a careful analysis of the synchronization subsystem.

This paper presents a methodology for validating the clock synchronization subsystem of ultrareliable computer systems. The validation methodology relies on the formal proof process to uncover design and coding errors and utilizes experimentation to validate the assumptions of the design proof. The methodology is demonstrated by application to the SIFT synchronization system.

The Validation Approach

Synchronization can be obtained in a distributed system by process-level protocols such as Ada's Rendezvous or by a clock

synchronization algorithm that maintains the system's distributed clocks within a specified skew. The latter approach was used in the SIFT computer.[2] This approach simplifies the design of the operating system and avoids the significant overhead associated with process-level synchronization.[3] The resulting system, however, is critically dependent upon the correct functioning of the clock synchronization subsystem. In a fault-tolerant system, the clock synchronization system must operate properly in the presence of faults.

Although the clocks are physically separated, clock failures are not independent, because during synchronization each clock uses other clock values. Because of this failure dependency, a fault-tolerant clock synchronization algorithm is used to prevent the propagation of a clock failure to another clock in the system. The validation process must establish the correctness of this algorithm in a system context. The following failure modes are considered: 1) an error exists in the design, 2) an error occurred in implementing the design, and 3) the design assumptions have been violated.

In this paper, a validation methodology considering each of these failure modes is introduced. This methodology is demonstrated by application to the SIFT clock synchronization system. The following is an outline of the validation method:

1) Formally prove a theorem characterizing the guaranteed maximum clock skew as a function of measurable system parameters.

2) Verify by formal proof that the components (hardware or software) implement the synchronization algorithm.

3) By experimental measurements and by reference to the manufacturers' specifications, obtain values for the parameters used in the design assumptions of step 1. Estimate the probability that the assumptions are violated.

4) Construct a Markov model of the system, including the failure probabilities of step 3 to calculate the probability of failure.

Traditionally, synchronization systems have been developed using engineering techniques without a formal mathematical analysis of the properties of the synchronization technique. Validation has been accomplished via life-testing. If the clocks remained synchronized throughout the test period, the design was assumed to be adequate. If any problems were encountered, the design was modified until it "worked." The methodology presented in this paper is unique in that it relies on formal proof of correctness to force designers to make explicit statements describing the assumptions and mathematical relationships implied by their design.

In accordance with step 1, the designers of the SIFT computer system derived a mathematically analyzable synchronization algorithm. Today's designer is not limited to this algorithm or forced to derive his own. Recently, several synchronization algorithms have been developed whose correctness has been established by proof.[4,5,6] Each of the theorems contained in these proofs expresses the maximum clock skew as a function of two basic parameters—the maximum clock read error and the maximum clock drift rate. The parameters are defined in the proofs as axioms; e.g., it is explicitly assumed that a nonfaulty processor can read another nonfaulty processor's clock with an error no greater than a specified maximum read error and that nonfaulty clocks cannot drift apart at a rate greater than a specified maximum drift rate.

The second step of the methodology demonstrates that an error did not occur while implementing the mathematical algorithm as a functioning system. This step also establishes relationships between high-level design parameters and system components. The third step verifies that the assumptions of the proof are not violated in the implementation. Although the fault-tolerant clock synchronization algorithms that have appeared in the literature vary significantly, they are all based on the assumption that the clock read error and clock drift rate can be bounded. If the correctness of these assumptions can be demonstrated experimentally, then the adequacy of the synchronization subsystem can be determined analytically. The theorems guarantee that the algorithm will maintain a degree of synchronization even in the presence of the arbitrary malicious behavior of the faulty clocks. All of the experimental testing performed during the validation is aimed at demonstrating that the clock read error is sufficiently small and that the drift rates of the system clocks are within the necessary bounds.

The final step of the methodology incorporates the uncertainties associated with experimental measurement into the reliability analysis. Since all of the fault-tolerant clock synchronization theorems in the literature are based on the assumption of a bounded clock read error, this inevitably requires an estimation of the probability that an assumed bound will be exceeded. Since clock reads are performed periodically in a system, this probability can be translated into a failure rate and included in the Markov model of the system. In the remaining sections of this paper, the outlined steps of the validation process will be illustrated in detail by applying this validation method to the SIFT synchronization subsystem.

Step 1—Design Proof

The first step of the validation methodology is to perform a design proof of the synchronization algorithm. The design proof produces a mathematical theorem that relates the performance of the synchronization algorithm to specific properties of the underlying system. During the SIFT design, a design proof was made of the SIFT synchronization algorithm.[7] In this section, the essential aspects of this design proof theory developed by SRI International are given.

It is convenient to define a clock as a function from real time t to clock time T: $C(t) = T$. Real time will be distinguished from clock time by the use of small letters for the former and capital letters for the latter. It is sometimes useful to use the inverse clock function $r(T) = C^{-1}(T) = t$. Using this inverse function, the concept of synchronization can be defined: two clocks r_p and r_q are synchronized to within δ of each other at time T if

$$|r_p(T) - r_q(T)| < \delta \tag{1}$$

Next, the notion of a nonfaulty clock is defined: a clock r is a nonfaulty clock if it is a monotonic, differentiable function of T and there exists a ρ such that

$$\left|\frac{dr}{dT}(T) - 1\right| < \rho/2 \tag{2}$$

Thus, the drift rate of a nonfaulty clock from real time is bounded by $\rho/2$.

The clocks in the system are corrected every R seconds. Using $T^{(i)}$ as the clock time at the beginning of the i^{th} interval, $T^{(i)} = T^{(0)} + iR$ and $R^{(i)} =$ the interval $[T^{(i)}, T^{(i+1)}]$.

For each such interval there is a new clock definition:

$$C^{(i+1)}(t) = C^{(i)}(t) + \Delta_p^{(i)} \tag{3}$$

where $\Delta_p^{(i)}$ is the i^{th} clock correction. Thus, the time base of the system is formally represented by a sequence of mathematical functions each applicable to a different interval of real time. Each function differs from its predecessor by a constant.

The clock synchronization algorithm requires that each processor exchange clock values with every other processor during the subinterval $S^{(i)} = [T^{(i+1)} - S, T^{(i+1)}]$, which is the last S seconds of the interval $R^{(i)}$. Since this clock value exchange is subject to error, it is necessary to introduce a notation and an axiom that characterizes this error: if processors p and q are nonfaulty, then p obtains a value Δ_{qp} such that

$$|e_{qp}(T)| < \varepsilon \tag{4}$$

where

$$e_{qp}(T) = r_p^{(i)}(T + \Delta_{qp}) - r_q^{(i)}(T) \tag{5}$$

for some time T during the synchronization interval. The value Δ_{qp} is thus processor p's perception of processor q's skew with respect to itself. Processor p's error in reading processor q's clock e_{qp} is bounded by ε.

The synchronization algorithm is as follows.

Algorithm: for all p,

$$C_p^{(i+1)} = C_p^{(i)} + \Delta_p$$

where

$$\Delta_p = (1/N) \sum_{r=1}^{N} \bar{\Delta}_{rp}$$

$$\text{if } r \neq p \text{ and } |\Delta_{rp}| < \Omega \text{ then } \bar{\Delta}_{rp} = \Delta_{rp}$$

$$\text{else } \bar{\Delta}_{rp} = 0 \tag{6}$$

The following theorem is a trivial adaptation of the theorem proven in Ref. 7. (In the last step of the proof, it is assumed that $\Omega \simeq \delta + \varepsilon$ in order to obtain a simpler form of the δ bound. The following formula is the form of the bound in Ref. 7 prior to utilizing the preceding assumption.) This form of the theorem enables one to study the effect of an erroneous choice of Ω on the level of synchronization attainable.

Theorem: if

$$3m < N$$

$$\delta \geq (\varepsilon + \rho S) + 2m\Omega/(N - m) + N\rho R/(N - m)$$

$$\delta \geq \delta_0 + \rho R$$

$$\delta \ll R$$

$$\delta \ll \varepsilon/\rho \tag{7}$$

where N is the number of clocks in the system, m is the number of faulty clocks, and δ_0 is the initial clock skew at $T^{(0)}$, then the Algorithm satisfies the following:

S1. If up to time $T^{(i+1)}$ processors p and q are nonfaulty, then for all T in

$$R^{(i)}, |r_p^{(i)}(T) - r_q^{(i)}(T)| < \delta \tag{8a}$$

S2. If processor p is nonfaulty up to time $T^{(i+1)}$, then

$$|r_p^{(i+1)}(T) - r_p^{(i)}(T)| < \Omega \tag{8b}$$

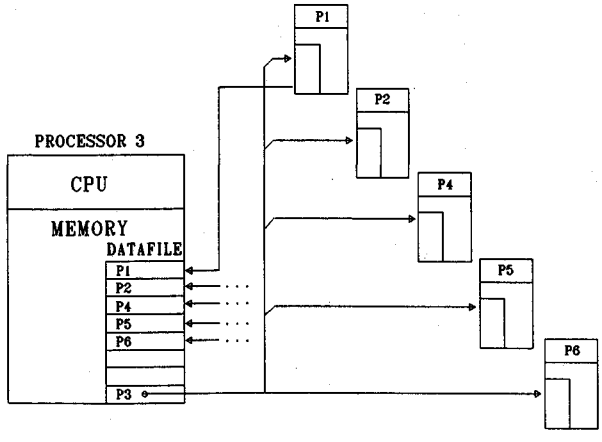


Fig. 1 SIFT system interconnect.

Statement (S1) of the theorem states that the maximum skew between any two nonfaulty clocks will be less than δ. The upper bound on δ is given in the theorem as a function of parameters N, m, ε, ρ, Ω, R, and S. Statement (S2) guarantees that the maximum correction will be less than Ω.

Step 2—Implementation Verification

During implementation verification, the synchronization system must be shown to implement the algorithm correctly, as described in the design proof. SRI International is currently verifying the SIFT implementation under NASA contract NAS1-17067. Although the details are not presented here, the connections between the design proof and the implementation are explained so that the later steps of the validation methodology can be presented in detail.

The SIFT Implementation

The Software Implemented Fault-Tolerance (SIFT) computer system was developed for NASA as an experimental vehicle for fault-tolerant systems research. The SIFT architecture consists of a fully distributed configuration of Bendix BDX930 processors. Messages are broadcast to all processors in the system over point-to-point links (Fig. 1). Although the design can accommodate up to eight processors, only six processors are in the current system. Hardware reliability estimations have demonstrated that this is adequate to meet the stated reliability goals of a probability of failure less than 10^{-9} for a 10-hour flight.

The important distinctions between SIFT and other fault-tolerant computers are:

1) The functions supporting fault tolerance are primarily implemented in software (e.g., voting).

2) Different tasks can be supported at different replication levels (i.e., a noncritical task may be simplex whereas more critical tasks can be replicated three- or five-fold).

3) The unit of reconfiguration is a complete computer, i.e., processor, memory, and busses.

4) The design is not based on a special CPU or memory design.

5) The redundant computers are loosely synchronized.

The SIFT operating system has two levels of authority. The Local Executive contains procedures that support scheduling, voting, and communications. The Global Executive consists of tasks that cooperate to prove synchronization, redundancy management (fault isolation and reconfiguration), and interactive consistency. The synchronization of the computers is fundamental to the correct functioning of the communication system. Interprocessor communication is completely asynchronous. No handshake signals or rendezvous mechanisms are used. The validity of the data is guaranteed by the precedence established in the task schedule and the synchronization of the processors.

The SIFT synchronization system is composed of a set of clocks with 1.6 μs resolution, SIFT's broadcast network, and a task that performs the synchronization algorithm. With SIFT's broadcast system it is impossible for a processor p to directly read the clock of another processor q. Instead, processor q reads its clock and transmits the clock value to processor p. Since the accuracy of processor p's perception of clock q depends on how quickly processor p recognizes the receipt of clock q's value, processor p executes a tight "wait-for loop" until clock q's value arrives. This read function must be repeated for every clock in the system. To accomplish the clock reads, the beginning of the synchronization task is divided into a sequence of "windows," one window per processor. During its transmit window, a processor repeatedly reads its clock and broadcasts its value. All other processors wait until either the clock value arrives or the window ends. When the clock value is received, processor p reads its clock. The skew is calculated as the difference between the two clocks minus the approximate communication delay. When all windows are completed, the

```
GLOBAL FUNCTION SYNCTASK: INTEGER;
CONST
   OMEGA = 40;                          (* ABOVE WHICH THE SKEW IS IGNORED *)
   COMMDELAY = 24;                      (* EXPECTED COMMUNICATIONS DELAY *)
   WINDOW = 71;

VAR
   I: PROCESSOR;                        (* 1..MAXPROCESSOR *)
   NUM,SUM,TERM,START_TIME: INTEGER;
   CLOCK_P,CLOCK_Q,DELTA: INTEGER;
   UNSEEN: ARRAY[PROCESSOR] OF BOOLEAN;
   SKEW: ARRAY[PROCESSOR] OF INTEGER;

BEGIN
   UNSEEN[PID] := FALSE;
   FOR I := MAXPROCESSORS DOWNTO 0 DO
      BEGIN
      SKEW[I] := 0;
      START_TIME := CLOCK;
      IF I = PID THEN                   (* SEND MY CLOCK *)
         REPEAT                         (* THE BROADCAST LOOP *)
            IF BROADCAST_DONE  THEN
               BROADCAST(CLOCK,CLOCK_BUFFER);
         UNTIL CLOCK - START_TIME > WINDOW
      ELSE IF WORKING[I] THEN           (* GET P_i'S CLOCK *)
         BEGIN
         UNSEEN[I] := TRUE;
         REPEAT                         (* THE RECEIVE LOOP *)
            CLOCK_P := CLOCK;
            IF BROADCAST_RECEIVED(CLOCK_BUFFER,I) THEN  (* COMPUTE SKEW *)
               BEGIN
               CLOCK_Q := GET_BROADCAST(CLOCK_BUFFER,I);
               SKEW[I] := CLOCK_P - CLOCK_Q - COMMDELAY;
               UNSEEN[I] := FALSE;
               REPEAT    (* WAIT UNTIL END OF WINDOW  *)
               UNTIL CLOCK - START_TIME > WINDOW;
               END;
         UNTIL CLOCK - START_TIME > WINDOW;
         END;

   SUM := 0;  NUM := 0;                 (* CALCULATE THE CLOCK CORRECTION. *)
   FOR I := 1 TO MAXPROCESSORS DO
      IF WORKING[I] THEN
         BEGIN
         TERM := SKEW[I];
         IF TERM >  OMEGA THEN TERM := 0;
         IF TERM < -OMEGA THEN TERM := 0;
         IF UNSEEN[I] THEN TERM := 0;
         SUM := SUM + TERM;  NUM := NUM + 1;
         END
   DELTA := SUM DIV NUM;
   CLOCK := CLOCK - DELTA;     (* ADJUST CLOCK *)
END; (* SYNCTASK *)
```

Fig. 2 SIFT synchronization task.

correction is calculated, and the clock is corrected. The Pascal code that implements the synchronization algorithm is shown in Fig. 2.

The Mapping of Design Parameters to the Implementation

The Mapping to Data Structures

An informal mapping between the design parameters and software data structures is given in Table 1. The relationships shown in Table 1 map the variables of the implementation to the design parameters except for the variable v, which does not appear in the design proof. This term results from the communication delay introduced into the clock read function as previously described, and is explained in the following section.

The Parameters ε, e_{qp}, and v

The following discussion relates the communication delay estimate v to e_{qp} and therefore to ε. As defined earlier, ε is an upper bound on the clock read error $|e_{qp}|$. Unfortunately, e_{qp} is defined in terms of real time rather than observable clock time. The following formula defining e_{qp} in terms of observable clock times can be shown to be a highly accurate approximation to the theoretical e_{qp}:[8]

$$A_{qp} + e_{qp} = \Delta_{qp} \tag{9}$$

where $A_{qp}(t)$ is the difference between clocks p and q at real time t [i.e., actual skew at t is $C_p(t) - C_q(t)$].

In the SIFT clock task, a process p reads a process q's clock by the following method: at a prespecified time t_1, process q reads its clock and transmits the value $C_q(t_1)$ to process p. Upon receiving the message at t_2, process p immediately reads its clock to obtain $C_p(t_2)$. If the exact communication delay X_{qp} [i.e., $C_q(t_2) - C_q(t_1)$] were known, then the exact skew could be calculated by

$$A_{qp}(t_2) = C_p(t_2) - C_q(t_1) - X_{qp} \tag{10}$$

The designer of the synchronization system chooses a value v (38.4 μs in SIFT) approximately equal to $E(X_{qp})$ to be used by the system to compute an apparent skew Δ_{qp} by the following formula:

$$\Delta_{qp}(t_2) = C_p(t_2) - C_q(t_1) - v \tag{11}$$

These computed values are used to calculate the clock correction value Δ_p according to the synchronization algorithm. However, because the communication delay is variable, Δ_{qp} differs from the actual skew A_{qp} by an error $e_{qp} = \Delta_{qp} - A_{qp} = X_{qp} - v$. There are two components to this error:

$$e_{qp} = X_{qp} - v = [X_{qp} - E(X_{qp})] + \mu \tag{12}$$

where $\mu = E(X_{qp}) - v$. The first component, $X_{qp} - E(X_{qp})$, is the variation due to the random nature of the communication. The second component μ is a bias due to the error in choosing v. Also, it follows from the preceding that $E(e_{qp}) = \mu$.

N, m, R, S, δ_0—Operating System Variables

The design parameters N, m, R, S, and δ_0 are related to operating system variables. The correct functioning of the synchronization algorithm is therefore dependent on the correctness of the underlying system process, such as task scheduling and redundancy management. We assume here that these processes have been validated and are correct.

The parameter N is the number of clocks in the system. ($N = 6$ in SIFT.) The first condition of the synchronization theorem relates the maximum number of faulty clocks the system can tolerate, m, to the total number of clocks in the system, N. Since the SIFT system can reconfigure, the values of N and m may change during system operation.

The design proof states that the algorithm must execute every R seconds and will complete in S seconds. In SIFT, the synchronization task is scheduled every major frame, $\simeq 105.6$ ms, and completes within one subframe, $\simeq 3.2$ ms.

The parameter δ_0 is the maximum initial skew between any two clocks in the system. The operating system executes a separate synchronization process upon system startup to achieve an initial synchronization to within δ_0. The value of δ_0 must be less than $\delta - \rho R$ to meet the requirements of the theorem.

The parameter Ω is defined in the algorithm as the largest acceptable skew. Any clock perceived to be skewed further than Ω is ignored. In SIFT, Ω is 200 μs.

The parameter ρ was defined for nonfaulty clocks by Eq. (2) as $\left|\frac{dr}{dT}(T) - 1\right| < \rho/2$. This parameter represents a specification requirement on the crystal oscillator, which is the time base of the processor's clock.

Validation Parameters

Certain system parameters not explicitly defined in the design proof must be considered in validating the system. These parameters reflect the system functions that depend upon system synchronization.

In order for the SIFT communication system to work properly, the prevote delay ψ must be greater than the maximum clock skew δ plus the maximum hardware broadcast time B:

$$\psi > \delta + B \tag{13}$$

Table 1 Mapping of theory to design

Theoretical notation	Data structure
Δ_{qp}	Skew [I]
$C_q(t_1)$	Clock Q
$C_p(t_2)$	Clock P
v	Commdelay
Ω	Omega
$\bar{\Delta}_{qp}$	Term
Δ_p	Delta

This constraint guarantees that all data is present before a vote begins. The prevote delay ψ is the minimum time between the start of a subframe and the initiation of voting (206.6 μs in SIFT). The maximum hardware broadcast time B in SIFT is 18.2 μs.

The length of the clock read window W is 320 μs. To guarantee a successful clock read, the window must be large enough to accommodate the maximum clock skew δ plus the maximum communication delay Max$[X_{qp}]$:

$$W > \delta + \text{Max}(X_{qp}) = \delta + \text{Max}(e_{qp} + v) = \delta + v + \varepsilon \quad (14)$$

Step 3—Experimental Validation of Assumptions

The design proof establishes the correctness of the algorithm under a set of assumptions. Many of the assumptions are well-established mathematical theorems. Others define the behavior of the computer system on which the algorithm executes. The design assumptions in the SRI design proof are:

1) The maximum drift rate between any two working clocks is $< \rho$.

2) A nonfaulty processor can read any other nonfaulty processor's clock to within an error ε.

3) The system executes the algorithm every R seconds and provides enough CPU time for the algorithm to complete, i.e., S seconds.

4) The clocks of the system are initially synchronized to within δ_0.

The probability that each design assumption will be violated during a mission of T hours must be estimated. These probabilities will be referred to as p_1, p_2, p_3, and p_4 respectively and will be estimated in the following subsections. In order to perform a Markovian reliability analysis, these probabilities will be converted to equivalent failure rates λ_1. λ_2, λ_3, and λ_4 respectively.

Design Assumption 1

Design assumption 1 defines the performance criteria for a nonfaulty clock in terms of the parameter ρ: $|dr/dT(T) - 1| < \rho/2$. The manufacturer's specification for the SIFT clocks is a drift rate less then 15 μs/s; therefore, $\rho/2 = 15$ μs/s. It is necessary to ensure that all clocks are working properly prior to the beginning of a mission. This is related to the problem of assuring that all processors are fault-free at time 0 (in the reliability model). Currently, diagnostic procedures do not provide 100% coverage of all gate faults (typically only 95% coverage is obtained). A quality control procedure is given in Appendix A which can be used to reduce the probability of including a faulty clock in the initial configuration to a value insignificant in comparison with the probability of including a processor with an undetected fault. The probability of clock failure after startup is included in the processor failure rate in the reliability analysis.

Design Assumption 2

This design assumption defines a worst-case error in reading another processor's clock in terms of the parameter ε:

$$|e_{qp}| < \varepsilon \quad (15)$$

Table 2 State dependency of read error bounds

(N, m)	ε_v, μs	ε_Ω, μs	ε_w, μs	ε, μs
(6, 0)	91.4	66.1	93.3	66.1
(6, 1)	51.2	39.4	66.6	39.4
(5, 0)	91.4	66.1	93.3	66.1
(5, 1)	41.2	32.7	59.9	32.7
(4, 0)	91.4	66.1	93.3	66.1
(4, 1)	24.5	21.5	48.7	21.5
(3, 0)	91.4	66.1	93.3	66.1
(1, 0)	91.4	66.1	93.3	66.1

Theoretically this parameter is an absolute bound. However, since e_{qp} is a random variable with an unknown distribution, it is impossible to select an upper bound with 100% confidence. The best that can be done is to estimate the probability that a bound will be exceeded. This probability, p_2, will be estimated in this section from experimental data obtained from the SIFT system. First, an experimental method for measuring e_{qp} is described. Second, a method for determining ε is given. Finally, the probability that this bound is exceeded is estimated using a statistical technique capable of dealing with extremely large quantiles.

Measuring e_{qp}

As shown earlier, $A_{qp} + e_{qp} = \Delta_{qp}$. Since Δ_{qp} is directly obtainable from the SIFT processor memories, e_{qp} can be readily calculated if A_{qp} is known. For our analysis, a global clock that can be read concurrently by all processors of the system was used to determine A_{qp}. An alternate method of measuring e_{qp} if a global clock is not available is presented in Appendix B.

The global clock is used as real time and thus provides a direct means of measuring actual skew. The clock synchronization task was instrumented so that each processor reads the global clock at clock time T obtaining a real-time value $r_p(T)$. The actual skew $a_{qp}(T)$ in real time is

$$a_{qp}(T) = r_p(T) - r_q(T) \quad (16)$$

It is easily shown that $A_{qp} \simeq a_{qp}$ to within a factor $\rho A_{qp} \simeq 10^{-9}$ s. The measurement of actual skew should be made at the same time as the Δ_{qp}'s are obtained. This, of course, is impossible. As long as A_{qp} is determined within a few hundred microseconds of obtaining the Δ_{qp}, the error is negligible (i.e. less than 10^{-9} s).

Determining ε

Before p_2 can be estimated, the value of ε must be established. Although ε is not explicitly defined in the system being validated, it is implicitly constrained in three places:

$$\psi > \delta + B \quad (13)$$

$$\Omega > \delta + \varepsilon \quad (17)$$

$$W > \delta + \text{Max}(X_{qp}) = \delta + v + \varepsilon \quad (14)$$

Substituting $\delta = 2(\varepsilon + \rho S) + 2m\Omega/(N - m) + N\rho R/(N - m)$ from the design proof theorem in each of the three given formulas and solving for ε yields the following three constraints for ε:

$$\varepsilon_v < 1/2[\psi - B - \rho(2S + NR/(N - m))] - m\Omega/(N - m) \quad (18)$$

$$\varepsilon_\Omega < (N - 3m)\Omega/3(N - m) - \rho[2S + NR/(N - m)]/3 \quad (19)$$

$$\varepsilon_w < 1/3[W - 2\rho S - 2m\Omega/(N - m) - N\rho R/(N - m) - v] \quad (20)$$

The largest ε the system design can accommodate in each configuration (N, m) is

$$\varepsilon(N, m) = \text{Min}[\varepsilon_v(N, m), \varepsilon_\Omega(N, m), \varepsilon_w(N, m)] \quad (21)$$

Since ε depends on N and m, the probability p_2 will be a function of the SIFT configuration, and consequently the SIFT synchronization failure analysis cannot be decoupled from the hardware failure analysis. The values in Table 2 were calculated using the preceding formulas.

Estimating Prob $[e_{qp} > \varepsilon]$

In this subsection, the probability $p_2(T) = \text{Prob}[$one or more read errors $> \varepsilon(N, m)$ occurring on a specific processor within the mission time $T]$ is calculated. This probability is closely

related to the probability p_ε:

$$p_\varepsilon = \text{Prob[read error} > \varepsilon \text{ for a single clock read]}$$

The number of reads performed by SIFT within time T is $\eta = (N-1)T/R$. Assuming that the clock reads are independent and letting $\lambda_2 = (\eta/T)p_\varepsilon$,

$$p_2(T) = 1 - [1 - (\lambda_2 T)/\eta]^\eta \tag{22}$$

For a large η, the Poisson approximation is valid:

$$p_2(T) \simeq 1 - e^{-\lambda_2 T} \tag{23}$$

Thus $p_2(T)$ is approximately an exponential distribution with a failure rate λ_2.

The maximum observed e_{qp} is 18.2 μs. For $m = 0$, ε is on the order of 66.1 μs—well beyond any observed data value for reasonable sample sizes. Therefore, estimation in the tail of the distribution is necessary. (If exceedingly large samples, e.g., 10^9 observations, are obtained or the system is poorly tuned such that some of the sample read errors exceed the bound, then the classical approach to quantile estimation can be used.)

A statistical method was developed by Bruce Hill for the estimation of large quantiles of a distribution.[9] The estimation is derived from the largest observations of a random sample. Hill's method does not require the assumption of any global form for the entire distribution function. Rather, inferences about the behavior in the tail of the distribution are based on the assumption that the tail conforms to the Zipf form. Many common distributions reduce to the Zipf form in the tails. If the k largest observations of the random sample have a distribution of the form

$$Z(y) = 1 - Cy^{-\alpha} \tag{24}$$

then this part of the tail is of the Zipf form. The maximum likelihood estimators for parameters α and C describing the behavior in the tail are

$$\hat{\alpha} = k/\left[\left(\sum_{i=1}^{k-1} \ell n\, y_{(i)}\right) - (k-1)\ell n\, y_{(k)}\right] \text{ and} \tag{25}$$

$$\hat{C} = (y_{(k)})^{\hat{\alpha}}\, k/N \tag{26}$$

where the k largest order statistics of the random sample are denoted by $y_{(1)} \geq y_{(2)} \geq \ldots \geq y_{(k)}$.

If the k extreme order statistics are of the Zipf form, then the set of V_i's

$$V_i = i\,\ell n[y_{(i)}/y_{(i+1)}] \text{ for } y = 1 \text{ to } k-1 \tag{27}$$

are independent, identically distributed exponential random variables with parameter α. Thus, the portion of the tail that is of the Zipf form may be determined by selecting the largest k for which the V_i's are found to be exponentially distributed by a standard test, such as the Chi-square goodness-of-fit test.

Table 3 Probability of failure and failure rate

(N, m)	ε (μs)	p_ε	$\lambda_2(N, m)$
(6, 0)	66.1	0.48×10^{-13}	0.83×10^{-8}
(6, 1)	39.4	0.87×10^{-9}	0.15×10^{-3}
(5, 0)	66.1	0.48×10^{-13}	0.66×10^{-8}
(5, 1)	32.7	0.29×10^{-7}	0.41×10^{-2}
(4, 0)	66.1	0.48×10^{-13}	0.50×10^{-8}
(4, 1)	21.5	0.79×10^{-4}	0.82×10^{1}
(3, 0)	66.1	0.48×10^{-13}	0.33×10^{-8}

Application of Method to SIFT e_{qp} Data

Using the global clock method, e_{qp} histograms were obtained for all pairs of the six SIFT processors. The Chi-square goodness-of-fit test was used to determine the size of the tail of each sample population that was of the Zipf form, and this portion of the tail was used to estimate the p_ε quantile. Some portion of each sample population tail conformed to the Zipf form, so Hill's method was applicable to every sample. The probability that a read error e_{qp} will exceed ε is expressed as

$$p_\varepsilon = P[e_{qp} > \varepsilon] = 1 - P[e_{qp} \leq \varepsilon] = 1 - Z(\varepsilon) = C\varepsilon^{-\alpha} \tag{28}$$

Using this technique on each q, p pair, the probabilities of failure were calculated for each $\varepsilon(N, m)$. The worst case was observed for processor pair 3, 7. The e_{qp} distribution for processor pair 3, 7 is shown in Fig. 3. Since the system can perform no worse than if each processor pair exhibited the behavior of processor pair 3, 7, this data was used to calculate the failure rates used in the reliability model. For processor pair 3, 7, $\alpha = 0.189 \times 10^2$ and $C = 0.1212 \times 10^{22}$. Table 3 contains the probabilities of failure and the failure rates derived.

Design Assumption 3

The design proof assumes that every R (clock) seconds each nonfaulty processor executes the synchronization algorithm. Since SIFT uses a static schedule table, the validity of this assumption depends primarily on the correctness of the scheduling table and the dispatching code. Such issues can be addressed using formal code and design proof techniques. For the purposes of this paper, the correctness of the scheduler is assumed, and consequently $p_3 = 0$.

Design Assumption 4

Design assumption 4 requires that a specific level of synchronization be attained during system startup. This process is performed before the system is brought on line. The SIFT system typically attains an initial synchronization of 10 μs, which is less than the $\delta_0 = 129.8$ required by the theorem. This condition of initial synchronization is easily tested at system startup, and thus we assume that $p_4 = 0$.

Step 4—Markovian Analysis of System Reliability

In the preceding section, the probability that each design axiom may be violated and the corresponding failure rates were determined. The first and last probabilities p_1 and p_4 affect the

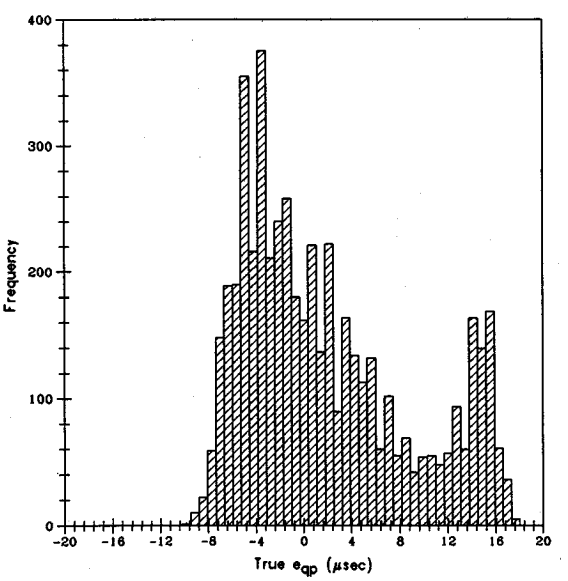


Fig. 3 Histogram of 5100 e_{qp}'s.

initial conditions of the reliability model. Although the system's reliability depends fundamentally on perfect maintenance between missions, current circuit testing techniques can provide only about 99% coverage of gate-level faults. Nevertheless, traditional analyses of fault-tolerant architectures have assumed perfect maintenance. For simplicity, we will likewise assume that the probability of being in the initial fault-free state at time 0 is 1. In the previous sections we have illustrated how the initial state of the clocks can be tested to a much higher level of confidence than the rest of the processor.

The other two failure probabilities correspond to failures during the mission. In this paper we are only considering the first probability p_2, the probability of an excessive read error. As mentioned before, the reliability of the synchronization subsystem cannot be decoupled from the analysis of processor failure (since synchronization failure probability depends on N and m). Therefore, the overall SIFT fault tolerance strategy must be modeled as well as the clock synchronization subsystem. For simplicity, only permanent processor failures will be included in the analysis. The failure rate of the processors is assumed to be $\lambda_p = 10^{-4}$.

In the case of SIFT, the clock is internal to the CPU, but not all processor failures disable the processor's ability to broadcast its clock. For the purposes of the reliability analysis, the fraction of hardware faults that inhibit correct clock function is denoted as γ, and these faults will be referred to as clock faults. Therefore, the clock failure rate is $\gamma\lambda_p$. The failure rate due to excessive read errors is λ_2 and is a function of N and m. The reliability model includes the following modes of SIFT: 1) the number of active processors ≤ 2 times the number of active faulty processors, and 2) the number of active processors ≤ 3 times the number of faulty clocks plus read failures.

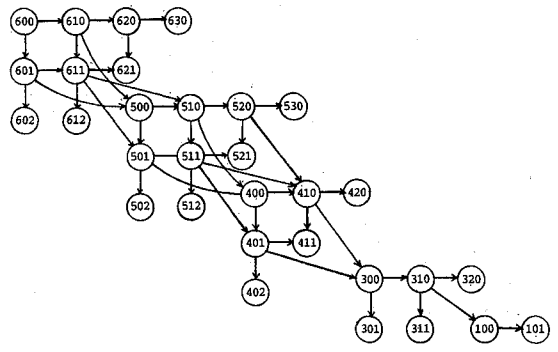
SIFT Reliability Model

The Markov model shown in Fig. 4 was constructed to describe the behavior of the SIFT system. The states of the model are denoted as (N, f, m) where N = number of processors in the configuration (1–6, excluding 2), f = number of processors with faults not affecting their clocks (0–3), and m = number of processors with clock faults (0–2). For example, state (6, 1, 2) has six processors in the configuration, of which two have clock faults and one has a fault that does not affect its clock.

Death States

The following define the death states of the model:

- if $[(N = 5)$ or $(N = 6)]$ and $[(m > 1)$ or $(f + m > 2)]$
- if $(N = 4)$ and or $(f + m > 1)$
- if $(N = 3)$ and $[(m > 0)$ or $(f > 1)]$
- if $(N = 1)$ and $(m + f > 0)$



STATE TRANSITION RATES:

600,610 $6(1-\gamma)\lambda_p$	600,601 $6\lambda_2(6,0)+6\gamma\lambda_p$	610,620 $5(1-\gamma)\lambda_p$	610,611 $5\lambda_2(6,0)+5\gamma\lambda_p$
610,500 ϕ_p	620,630 $4(1-\gamma)\lambda_p$	620,621 $4\lambda_2(6,0)+4\gamma\lambda_p$	620,510 $2\phi_p$
601,611 $5(1-\gamma)\lambda_p$	601,602 $5\lambda_2(6,1)+5\gamma\lambda_p$	601,500 ϕ_c	611,612 $4\lambda_2(6,1)+4\gamma\lambda_p$
611,621 $4(1-\gamma)\lambda_p$	611,510 ϕ_p	611,501 ϕ_c	500,501 $5\lambda_2(5,0)+5\gamma\lambda_p$
500,510 $5(1-\gamma)\lambda_p$	510,511 $4\lambda_2(5,0)+4\gamma\lambda_p$	510,520 $4(1-\gamma)\lambda_p$	510,400 ϕ_p
520,521 $3\lambda_2(5,0)+3\gamma\lambda_p$	520,530 $3(1-\gamma)\lambda_p$	520,410 $2\phi_p$	501,502 $4\lambda_2(5,1)+4\gamma\lambda_p$
501,511 $4(1-\gamma)\lambda_p$	501,400 ϕ_c	511,512 $3\lambda_2(5,1)+3\gamma\lambda_p$	511,521 $3(1-\gamma)\lambda_p$
511,401 ϕ_p	511,410 ϕ_c	400,401 $4\lambda_2(4,0)+4\gamma\lambda_p$	400,410 $4(1-\gamma)\lambda_p$
410,411 $3\lambda_2(4,0)+3\gamma\lambda_p$	410,420 $3(1-\gamma)\lambda_p$	410,300 ϕ_p	401,402 $3\lambda_2(4,1)+3\gamma\lambda_p$
401,411 $3(1-\gamma)\lambda_p$	401,300 ϕ_c	300,301 $3\lambda_2(3,0)+3\gamma\lambda_p$	300,310 $3(1-\gamma)\lambda_p$
310,311 $2\lambda_2(3,0)+2\gamma\lambda_p$	310,320 $2(1-\gamma)\lambda_p$	310,100 ϕ_p	100,101 λ_p

Fig. 4 Markov model of SIFT system reliability.

Transitions

From each nondeath state with $N > 0$ there are two failure transitions with rates $(N - f - m)(1 - \gamma)\lambda_p$ and $(N - m) \times [\gamma\lambda_p + \lambda_2(N, m)]$. Two recovery transitions with rates ϕ_p and ϕ_c characterize the SIFT reconfiguration process. From each nondeath state with $f > 0$ there is a recovery transition leaving it with rate $f * \phi_p$. From each nondeath state with $m = 1$ there is a recovery transition leaving it with rate ϕ_c.

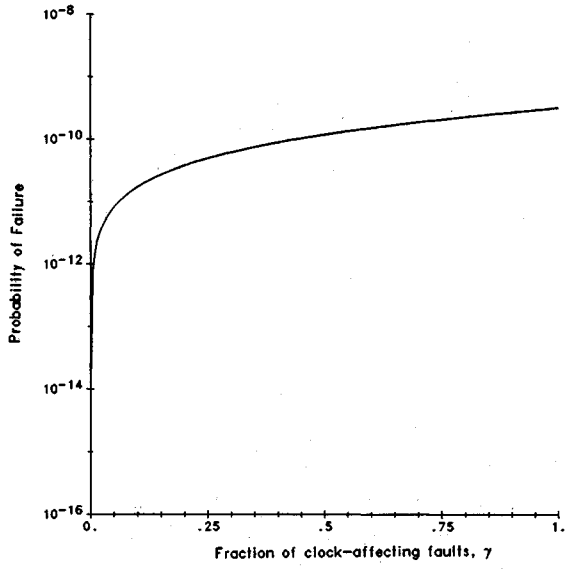


Fig. 5 Unreliability vs fraction clock faults.

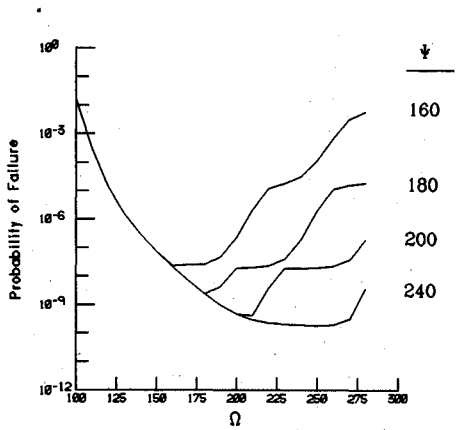


Fig. 6 Unreliability as a function of ψ and Ω.

Table 4 Optimal values of Ω and W for various values of ψ

ψ	Ω	W	P_f
120	126	170	5.9×10^{-6}
140	151	190	4.4×10^{-7}
160	161	200	2.3×10^{-8}
180	184	230	2.0×10^{-9}
200	207	250	3.7×10^{-10}
220	229	270	2.1×10^{-10}
240	252	300	1.9×10^{-10}
260	275	320	1.8×10^{-10}
280	282	330	1.8×10^{-10}

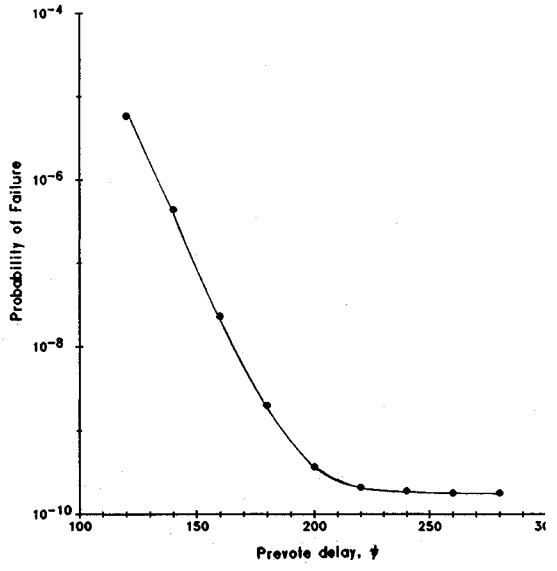


Fig. 7 System reliability as a function of ψ.

System Failure Analysis

The recovery rates have not been measured. For the purpose of analysis, a recovery rate of 1.8×10^4/h will be used for both ϕ_c and ϕ_p. This corresponds to recovery within two SIFT scheduling frames. The proportion of processor failures that affect the clock, γ, cannot be easily determined. Figure 5 shows the dependence of the system reliability on γ. Under the worst-case assumption of $\gamma = 1$, the probability of failure for the SIFT system is 4.7×10^{-10}.

The methodology of this paper can be used by a system designer to determine the optimal parameter values for a synchronization system. By evaluating the reliability model as a function of the "tunable" synchronization parameters, the parameter values yielding maximum reliability can be found. Figure 6 shows the SIFT system reliability as a function of ψ and Ω. Using such plots, the values of Ω and W yielding maximum reliability given the SIFT value of ψ, 202.6 μs, and $\gamma = 1$ were 210 and 250 μs respectively. The optimal values of Ω and W were found to be relatively insensitive to γ.

The parameter ψ is the synchronization system's impact on the performance of the SIFT operating system; the larger the ψ, the longer the operating system must wait before voting. Consequently the system designer must consider the tradeoff between system reliability and performance. For each value of ψ, the optimal values of Ω and W differ. Table 4 shows the optimal values of Ω and W for several values of ψ. Figure 7 shows the system reliability as a function of ψ and the corresponding optimal Ω and W parameters.

Conclusions

The validation method presented in this paper exploits the precision with which the formal proof method reduces the complexity of the system to verifiable axioms about the system behavior. The validation method introduced in this paper is essentially to 1) perform a design proof of the synchronization algorithm under the assumption of low-level system behavior axioms, 2) perform a code proof of the synchronization code, 3) experimentally estimate the probability that the system behavior axioms will be violated, and 4) include these failure probabilities into a reliability analysis of the system. The methodology is demonstrated by validating the SIFT system.

This method relies fundamentally on the development of a theorem that relates the maximum clock skew in terms of the clock read error and other observable system parameters. Many such theorems have appeared in the literature describing the behavior of many varied synchronization algorithms. This validation methodology could be applied to any system that is based on one of these algorithms. It is not necessary for fault-tolerant system designers to invent new algorithms and perform elaborate mathematical proofs, since such work is readily available. If a synchronization system is developed using ad hoc techniques, the development of such a theorem may be impossible or prohibitively difficult. This methodology could not be used to validate such a system. However, since it is no more difficult to build a system using one of these provably correct algorithms than using an algorithm based on ad hoc techniques, it is recommended that future system builders make use of the many existing algorithms. System designers can concentrate on methods of efficiently implementing these existing algorithms and the validation process.

Appendix A: Quality Assurance Test to Ensure Clocks are Within Specification

There are several techniques available to measure the drift rate of a clock crystal. A standard-frequency testing device could be used, or the relative drift rates between the clocks can be determined using the "endrifting method" described in Appendix B. The objective is to estimate an upper bound of all clock drift rates such that the probability that this bound is exceeded (due to measurement error) is negligible in comparison to the hardware failure probability of the device.

It is necessary to calculate Prob$[\rho_{qp} > \rho]$ for some q and p where ρ_{qp} represents the drift rate between clocks p and q. If there are n processors in the system being validated, then there are $n_c = \binom{n}{2}$ drift rates between processor pairs. For simplicity, these will be referred to as ρ_i, $i = 1$, and n_c. Using the linear regression analysis on the $\Delta_{qp}(T^{(i)})$ data described previously, a set of estimates

$$\{(\rho_i, \sigma_i^2) | i = 1, n_c\}$$

can be obtained, where ρ_i is an estimate of the drift rate between processor pair i and σ_i^2 is an estimate of the variance of ρ_i.

From the experimental data, an estimate of the upper bound of the drift rates ρ must be determined such that $\alpha = \text{Prob}[\text{Max}(\rho_i) > \rho]$ is sufficiently small. The following estimate is easily shown to be adequate:

$$\rho = \text{Max}(u_i) \tag{A1}$$

where u_i is defined by

$$\text{Prob}[\rho_i > u_i] = (1 - \alpha)^{1/n_c}$$

Theorem: Prob$[\text{Max}(\rho_i) < \rho] \geq 1 - \alpha$.
Proof:

$$\begin{aligned}
&\text{Prob}[\text{Max}(\rho_i) < \rho] \\
&\quad = \text{Prob}[\text{Max}(\rho_i) < \text{Max}(u_i)] \\
&\quad = \text{Prob}[\rho_1 < \text{Max}(u_i) \wedge \rho_2 < \text{Max}(u_i) \wedge \ldots \wedge \rho_n < \text{Max}(u_i)] \\
&\quad \geq \text{Prob}[\rho_1 < u_1 \wedge \rho_2 < u_2 \wedge \ldots \wedge \rho_n < u_n] \\
&\quad = \prod_{i=1}^{n_c} (1 - \alpha)^{1/n_c} \\
&\quad = 1 - \alpha
\end{aligned}$$

Endproof.
The u_i's are easily obtained by use of the following formula:

$$u_i = \rho_i + t(v, \theta)\sigma_i \tag{A2}$$

Table 5 Relative drift rates and corresponding upper bounds

Processor pair	i	ρ_i	σ_i	u_i
12	1	23.66	2.001×10^{-2}	23.77
13	2	4.034	2.131×10^{-2}	4.145
14	3	19.93	2.054×10^{-2}	20.04
15	4	1.278	2.203×10^{-2}	1.393
16	5	16.49	2.067×10^{-2}	16.60
23	6	19.61	1.842×10^{-2}	19.71
24	7	3.750	1.887×10^{-2}	3.848
25	8	24.96	2.226×10^{-2}	25.08
26	9	7.154	1.791×10^{-2}	7.248
34	10	15.86	1.738×10^{-2}	15.95
35	11	5.364	1.716×10^{-2}	5.453
36	12	12.45	1.483×10^{-2}	12.53
45	13	21.20	1.646×10^{-2}	21.28
46	14	3.415	1.450×10^{-2}	3.490
56	15	17.80	1.130×10^{-2}	17.86

where $v = n_s - 2$, n_s = number of data points used in the regression analysis to obtain ρ_i and σ_i, $\theta = (1-\alpha)^{1/n_c}$, and $t(v, \theta) = \theta$ percentage point of student's t distribution with v degrees of freedom.

For $n_s > 100$, $t(v, \theta)$ may be replaced by a percentage point of the Standard Normal distribution. The following approximation formula for the Normal distribution $F(z)$ is useful for small α (large z):

$$1 - \alpha = F(z) = 1 - (1/z\ 2\pi)\ \mathrm{Exp}[-(1/2)z^2] \qquad \text{(A3)}$$

A regression analysis of the $\Delta_{qp}(T^{(i)})$ data for each processor pair produced the values shown in Table 5. The u_i's were calculated for $\alpha = 10^{-7}$, $n_s = 510$. Thus $t(v, \theta) = 5.2056$. The maximum drift rate ρ is chosen as the maximum u_i; thus $\rho = 25.08\ \mu$s/s and $p_1 = 10^{-7}$, which is small relative to p_h.

This value of ρ is specific to the particular clocks in the SIFT implementation. Another set of six clocks is likely to have a different value. Therefore, the manufacturer's specification of $\rho = 30\ \mu$s/s was used in the reliability analysis in the body of the paper. The method of this appendix should be applied to every implementation to ensure that the clocks are within specification.

Appendix B: Alternate Method of Measuring e_{qp}

In this appendix, an alternate method of measuring e_{qp} which we call the "endrifting method" is described. This method does not require a special global clock but does rely on a few additional assumptions.

The SIFT clocks continually "correct" themselves during the synchronization task and thus exhibit periodic discontinuities. If the clock correction is disabled in order to sample the uncorrected clocks, the $\Delta_{qp}^{(i)}$'s exhibit a step function relationship to the iteration counter i (see Fig. 8). This unexpected behavior is due to the broadcast and receive loops used to transfer clock readings during each window of the synchronization task. The steps are created as two processors drifting past each other and complete the clock read during different cycles in the loop. The real drift rate is still linear, and the "noise" obscuring the line is the read error e_{qp}. Hence a linear least-squares analysis could be used to obtain $A_{qp}^{(i)}$ and consequently e_{qp}. There are two problems with this method:

1) With clock correction disabled, the SIFT system rapidly loses synchronization, and thus only small amounts of continuous data can be obtained at one time.

2) The $e_{qp}^{(i)}$'s are being measured in a nonrealistic situation. Perhaps they are statistically different when the system is synchronized, e.g., if there is a coupling between the communication system delays and the skew between the communicating processors.

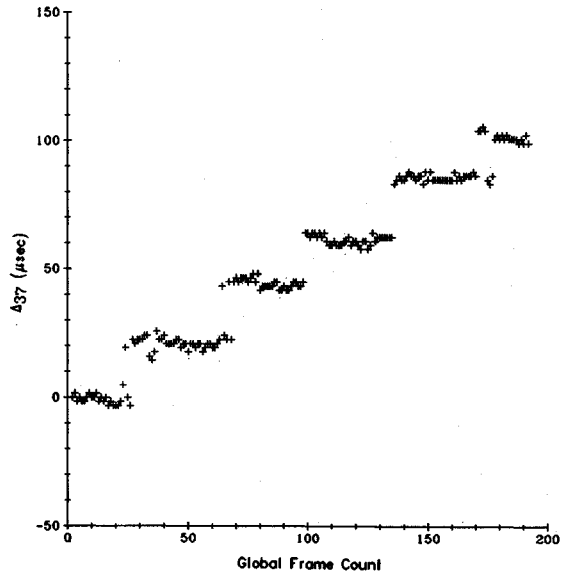


Fig. 8 Uncorrected clock's Δ_{qp}.

In the endrifting method, the synchronized clocks are allowed to run for a number of synchronization intervals, keeping track of the correction $\Delta_p^{(i)}$ made to each clock at each interval. Using the clock corrections, the Δ_{qp} values can be converted back to a common linear clock by a method we call "endrifting." This method uses the following data directly available from the SIFT processor memories:

$\Delta_{qp}^{(i)}$ for all pairs of processors p and q and $1 \leq i \leq n$ for some n (p's perception of its skew relative to q during iteration i), and

$\Delta_p^{(i)}$ for all processors p and $1 \leq i \leq n$ for some n (the calculated correction that p adds to its clock during iteration i).

From this data we obtain A_{qp} using the fact that unsynchronized clocks, although not identical, are linear functions of time.

In the design proof theory, the concept of "correcting" a clock was shown to be equivalent to defining a new clock function in terms of the previous clock. Thus the i^{th} iteration clock function is defined in terms of the $(i-1)^{\text{th}}$ clock by the following formula:

$$C_p^{(i)}(t) = C_p^{(i-1)}(t) + \Delta_p^{(i-1)} \qquad \text{(A4)}$$

Consequently,

$$C_p^{(i)} = C_p^{(0)} + \sum_{j=1}^{i-1} \Delta_p^{(j)} \qquad \text{(A5)}$$

The i^{th} clock function can thus be converted to equivalent $C_p^{(0)}$ clock values. Now since

$$\Delta_{qp}^{(i)}(t_2) = C_p^{(i)}(t_2) - C_q^{(i)}(t_1) + v \qquad \text{(A6)}$$

we have

$$\Delta_{qp}^{(0)}(t_2) = C_p^{(0)}(t_2) - C_q^{(0)}(t_1) + v \qquad \text{(A7)}$$

where $\Delta_{qp}^{(0)}(t_2)$ is the apparent skew computed with uncorrected clocks. Writing this in terms of the clock functions during iteration i gives

$$\begin{aligned}\Delta_{qp}^{(0)}(t_2) &= \left[C_p^{(i)}(t_2) - \sum \Delta_p\right] - \left[C_q^{(i)}(t_1) - \sum \Delta_q\right] + v \\ &= [C_p^{(i)}(t_2) - C_q^{(i)}(t_1) + v] - \sum \Delta_p + \sum \Delta_q \\ &= \Delta_{qp}^{(i)}(t_2) - \sum \Delta_p + \sum \Delta_q \qquad \text{(A8)}\end{aligned}$$

where $T_2 = C_p^{(i)}(t_2)$ and $T_1 = C_q^{(i)}(t_1)$ are in the interval $[T^{(i)}, T^{(i+1)}]$.

The endrifted $\Delta_{qp}^{(0)}$'s represent the skew that would have been present between processors q and p if the clocks had not been corrected. The endrifted $\Delta_{qp}^{(0)}$'s are thus a linear function of real time. The following model describes the system:

$$\Delta_{qp}^{(0)}(t) = \delta_{qp0} + \rho_{qp}t + e_{qp}(t) \tag{A9}$$

where δ_{qp0} is the initial skew between clocks q and p and ρ_{qp} is the drift rate between clocks q and p.

Performing a least-squares fit to the endrifted data, parameters α, β, and ϕ can be found where α and β are the y intercept and slope respectively and $\phi(t)$ is the set of residuals. The preceding equation may then be rewritten as

$$\Delta_{qp}^{(0)}(t) = \alpha + \beta t + \phi(t) \tag{A10}$$

If the e_{qp}'s are distributed with mean zero, then β is an estimate of ρ_{qp} and the residuals have approximately the same distribution as the e_{qp}'s. However, this may not be the case. Suppose that $E(e_{qp}) = \mu \neq 0$. Then $\alpha = \delta_{qp0} + \mu$, and the residuals have approximately the same distribution as $e_{qp} - \mu$.

Consequently,

$$e_{qp}(t) = \mu + \Delta_{qp}^{(0)}(t) - \alpha + \beta t \tag{A11}$$

The bias μ can easily be estimated as follows: from

$$A_{qp} + e_{qp} = \Delta_{qp} \tag{A12}$$

we have

$$A_{qp} + A_{pq} + e_{qp} + e_{pq} = \Delta_{qp} + \Delta_{pq} \tag{A13}$$

Furthermore, since $A_{qp} = -A_{pq}$ by definition

$$\Delta_{qp} + \Delta_{pq} = e_{qp} + e_{pq} \tag{A14}$$

Thus, under the assumption that the clock read process is symmetrical and hence $E(e_{qp}) = E(e_{pq})$, we have

$$\mu = E(e_{qp}) = E([\Delta_{qp} + \Delta_{pq}]/2) \tag{A15}$$

Therefore, μ can be estimated by the sample average of several observations of $[\Delta_{qp} + \Delta_{pq}]/2$.

References

[1]Wensley, J. H., "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," *Proceedings of the IEEE*, Vol. 66, 1978, pp. 1240–1252.

[2]Smith, T. B., "Fault-Tolerant Clocking System," *Digest of FTCS*, Vol. 11, 1981, pp. 262–264.

[3]Lamport, L. "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems," *ACM Transactions on Programming Languages and Systems*, April 1984, pp. 254–280.

[4]Lundelius, J. and Lynch, N., "A New Fault-Tolerant Algorithm for Clock Synchronization," ACM Conference on Principles of Distributed Computing, 1984.

[5]Halpern, J. Y., Simmons, B., Strong, R., and Dolev, D., "Fault-Tolerant Clock Synchronization," ACM Conference on Principles of Distributed Computing, 1984.

[6]Krishna, C. M., Shin, G., and Butler, R. W., "Ensuring Fault-Tolerance of Phase-Locked Clocks," *IEEE Transactions on Computers*, Vol. C-34, Aug. 1985.

[7]Lamport, L. and Melliar-Smith, M., "Synchronizing Clocks in the Presence of Faults," *Journal of the ACM*, Vol. 32, Jan. 1985, pp. 52–78.

[8]Butler, R. W. and Johnson, S. C., "Validation of a Fault-Tolerant Clock Synchronization System," 1984, NASA TP-2346.

[9]Hill, B. M., "A Simple General Approach to Inference About the Tail of a Distribution," *The Annals of Statistics*, Vol. 3, 1975, pp. 1163–1174.