NASA Technical Memorandum 86322

Measurement of SIFT Operating System Overhead

Daniel L. Palumbo and Ricky W. Butler

APRIL 1985

NASA

NASA Technical Memorandum 86322

Measurement of SIFT Operating System Overhead

Daniel L. Palumbo and Ricky W. Butler
Langley Research Center
Hampton, Virginia

National Aeronautics
and Space Administration

**Scientific and Technical
Information Branch**

1985

Contents

Summary

The Software Implemented Fault Tolerance (SIFT) computer system was developed for NASA by SRI International as an experimental vehicle for fault-tolerant systems research. SIFT was delivered to the Langley Avionics Integration Research Laboratory (AIRLAB) in April 1982. Development and testing have continued at the NASA Langley Research Center, and several versions of the operating system have evolved. Each new version represents the different strategies employed to improve the performance of particular functions of the operating system. The three versions discussed in this paper are

Version B	as delivered (baseline)
Version R	improved reconfiguration performance
Version V	improved vote and reconfiguration performance

The SIFT operating system is a fully distributed, real-time executive with no master controller. The operating system is implemented in Pascal and performs the following major functions:

1. Periodic task scheduling and dispatching
2. Data communication and voting
3. Clock synchronization
4. Fault isolation
5. Reconfiguration
6. Interactive consistency

These operating system functions fall into two categories: Local Executive and Global Executive. The Local Executive performs functions local to an individual processor, i.e., (1) and (2) above. The Global Executive is a set of tasks which assume the responsibility of items (3) through (6).

The SIFT operating system utilizes significant resources to achieve fault tolerance in software. This overhead falls in two main categories:

1. The time required to vote the intertask communication variables at the beginning of each subframe
2. The time utilized by the executive tasks

The overhead from the first category, vote overhead, was found to be a linear function of the amount of data to be voted. The following table gives a basic comparison of the vote times (best values) for the versions of SIFT investigated with a six-processor configuration:

Version	Three-way vote time per buffer, ms	Five-way vote time per buffer, ms
B	0.413	0.412
R	.302	.357
V	.079	.107

The vote times were found to vary with the number of processors in the configuration and the location of the task replicates in the schedule table. These variations were typically on the order of 10 to 25 percent.

The overhead due to the second category, the executive task overhead, is given below.

Version	Overhead, ms	Frame size, ms	Overhead, percent of frame size
B	60.8	83.2	73.2
R	32.0	51.2	62.5
V	28.8	44.8	64.3

The reduced major frame size for Versions R and V arises because the improved vote performance enables some tasks to be scheduled in fewer subframes.

The SIFT computer system requires significant overhead to achieve its fault tolerance. The voting and interactive consistency functions were found to be the primary sources of operating system overhead. Unfortunately, these functions seem to be inherently expensive when implemented in software.

Introduction

The Software Implemented Fault Tolerance (SIFT) computer system was developed for NASA by SRI International as an experimental vehicle for fault-tolerant systems research. The SIFT effort began with broad, in-depth studies stating the reliability and processing requirements for digital computers which would, in the next generation of aircraft, control flight-critical functions. (See refs. 1 and 2.) Detailed design studies were made of fault-tolerant architectures which could meet the required reliability and processing requirements. (See refs. 3 and 4.) Following these studies, SRI International and the Bendix Corporation designed and built the SIFT system, which was delivered to the Langley Avionics Integration Research Laboratory (AIRLAB) in April 1982. The SIFT architecture consists of a fully distributed configuration of Bendix BDX-930 processors with a point-to-point communication link between every pair of processors. (See fig. 1.) Although the design can accommodate up to eight processors,

only six processors are in the current system; reliability estimations have demonstrated that this is adequate to meet the stated goal of a probability of failure of less than 10^{-9} for a 10-hour flight.

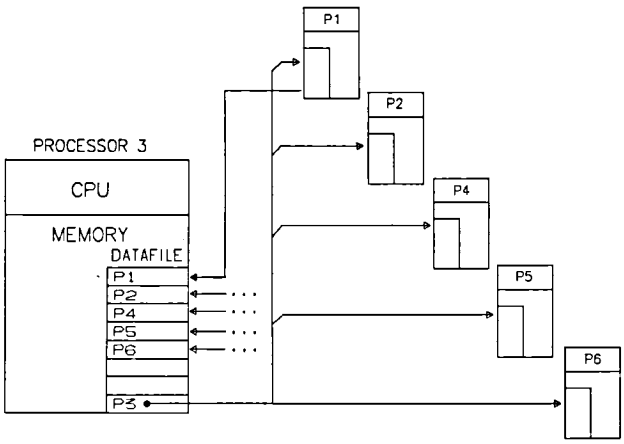

Figure 1. SIFT system interconnection.

The basic attributes of fault-tolerant computers are

1. Redundant hardware and tasks are used.
2. Errors caused by hardware faults are masked by voting the redundant outputs.
3. To increase reliability, faulty hardware is removed from the system by means of reconfiguration.

Important distinctions between SIFT and other fault-tolerant computers are

1. The functions supporting fault tolerance (e.g., voting) are primarily implemented in software.
2. Different tasks can be replicated to different levels (i.e., a noncritical task may be simplex, whereas more critical tasks can be replicated three-fold 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 central processing unit (CPU) or memory design.
5. The redundant computers are loosely synchronized.

The assignment of tasks to processors in SIFT is predetermined by a task schedule table, which is constructed by the application designer. The SIFT scheduler periodically dispatches tasks according to the task schedule. As processors fail, the hardware complement changes. Therefore, the application designer must define a task schedule for each level of configuration the system may encounter. Reconfiguration in SIFT is essentially accomplished by selecting the appropriate task schedule. The decision to reconfigure is based on error information gathered when the data from replicate tasks are voted.

The synchronization of the computers is fundamental to the correct functioning of the exact match vote algorithm and communication system. Synchronization insures that all replicates of a task receive the same input data and therefore produce the same output data if fault free. Interprocessor communication is completely asynchronous. No handshake signals or rendezvous mechanisms are used. The validity of data is guaranteed by the precedence established in the task schedule and the synchronization of the processors.

The SIFT operating system has two levels of authority. The Local Executive contains procedures which support scheduling, voting, and communications. The Global Executive consists of tasks which cooperate to provide synchronization and redundancy management (fault isolation and reconfiguration). Since the delivery of SIFT, development and testing have continued at the NASA Langley Research Center, and several versions of the operating system have evolved. Each new version represents the different strategies employed to improve the performance of particular functions of the operating system. The three versions discussed in this paper are

Version B as delivered (baseline)

Version R improved reconfiguration performance

Version V improved vote and reconfiguration performance

Version B only ran by disabling the clock interrupt during certain executive tasks. This was unacceptable because the disabling of interrupts resulted in significant delays in the output of the periodic application tasks. The primary problem was the large overhead of the Reconfiguration Task. Therefore, the operating system was redesigned. This new version is referred to as Version R. Version R is able to support reasonable task schedules without disabling the clock interrupts during certain executive tasks. Finally, the vote system was redesigned to improve the vote performance. This version is referred to as Version V. Version V was obtained by enhancement of Version R. Before the results of performance measurements on each version are discussed, an explanation of pertinent hardware and operating system internals is presented.

Hardware Configuration

The SIFT processors are Bendix BDX-930 avionics computers which communicate via a fully connected point-to-point broadcast network. Although the interconnection network and operating system

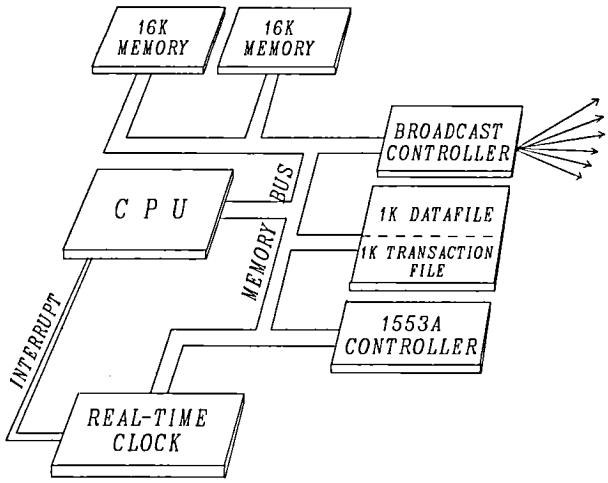

Figure 2. SIFT processor block diagram.

are able to support up to eight processors, only six processors are currently used in SIFT. Each computer in the system has a 16-bit CPU, 32K words of static random access memory (RAM), 1K datafile memory, 1K transaction file memory, a broadcast controller, a 1553A controller, and a real-time clock. (See fig. 2.) The CPU is constructed from four 2901 bit slice chips in a microprogrammed pipeline architecture and achieves a performance level of 1 million instructions per second. The 1553A controller provides a MIL-STD-1553A bus interface for communication with external aircraft systems.

Datafile

The datafile is a 1K memory block and serves as buffer area for the broadcast and 1553A controllers. The datafile is partitioned into eight 128-word sections. The first seven sections function as input "mailboxes." The other section serves as an output buffer. Each input data stream from the broadcast network is hardwired to a specific mailbox to maintain communication isolation.

To broadcast a value, the value is first stored in the datafile output section. To start the broadcast, the location of the value is loaded into the transaction pointer register. The broadcast transmitter signals completion after 14.7μs. This allows for worst-case contention for the receiving datafile.

Real-Time Clock

The real-time clock is a read/write register which produces interrupts at 1.6-ms intervals. The clock is a 16-bit counter that is driven by the 16-MHz crystal in the CPU. The clock is therefore synchronized exactly to the fetch-execute cycle of the CPU. The least significant bit of the clock has a value of 1.6 μs.

Operating System Overview

The SIFT operating system is a fully distributed, real-time executive with no master controller. The operating system is implemented in Pascal and performs the following major functions:

1. Periodic task scheduling and dispatching
2. Data communication and voting
3. Clock synchronization
4. Fault isolation
5. Reconfiguration
6. Interactive consistency

These operating system functions fall into two categories: Local Executive and Global Executive. The Local Executive performs functions local to an individual processor, i.e., (1) and (2) above. The Global Executive is a set of tasks which assume the responsibility of items (3) through (6). The major distinction between the Local and Global Executives is that the Global Executive tasks exchange data and cooperate on a systemwide basis, whereas the Local Executive procedures do not exchange data or cooperate.

The primary purpose of the SIFT operating system is to provide fault tolerance through masking of errors by voting of replicated data from the redundant tasks executing on separate computers. The voting is exact match (i.e., bit-by-bit comparison) and therefore requires that the replicated processes use exactly the same input data and produce their outputs prior to the vote. System coordination is achieved by use of a decentralized clock synchronization algorithm (ref. 4) and a simple communication protocol relying on the synchronization provided by this algorithm. Basically, the communication protocol requires that a data-producing task broadcast its data at some preagreed time T and that the data-receiving tasks wait until at least time T + Maximum broadcast time + Maximum clock skew before reading. Communication within SIFT is therefore critically dependent upon the correct performance of the clock synchronization algorithm. Data generated by a task are available to other tasks only after the termination of the data-producing task. The operating system allows the application system designer to define up to 128 "data buffers" for each processor's mailbox. Each of these "data buffers" consists of one word in the datafile area. An application task broadcasts its output data by calling a Local Executive procedure. The receiving task must be scheduled after termination of the last replicate of the data-producing task. Prior to execution of the receiving task, the Local Executive votes the replicates of the data and places the majority value in a "post vote"

array. To retrieve the voted data, the receiving task calls a Local Executive function. Several operating system data structures must be initialized by the application designer to control the functions which cooperate in performing the communications.

The Local Executive also keeps a count of every vote disagreement and the identity of the nonagreeing processors. This error information is distributed and analyzed by the Global Executive. From this analysis, the Global Executive decides whether a reconfiguration should take place. Although this function is transparent to the application tasks at runtime, the application designer must initialize several data structures to preplan the reconfiguration process.

Local Executive Data Structures

In this section, the functions performed by the Local Executive are described. Because the SIFT operating system is driven by static data structures, the descriptions center around these data structures.

The Local Executive has two main responsibilities: (1) scheduling and dispatching of tasks and (2) data communication and voting. The following data structures are used by the Local Executive:

1. Task schedule
2. Task table
3. Buffer information (BINF) array
4. Buffer table (BT) array
5. Vote schedule
6. POSTVOTE array

The data linkages between these structures are illustrated in figure 3. Only the POSTVOTE array

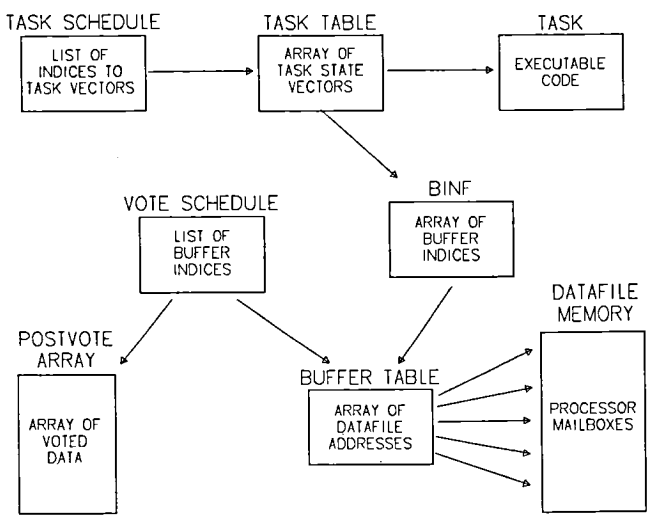

Figure 3. Linkages between SIFT data structures.

is constructed completely by the operating system. All the other structures require some, if not all, of their information to be entered by the application designer in BDX-930 assembly code.

Task schedule. Task scheduling in SIFT is nonpreemptive and based on precalculated schedule tables. The schedule table defines the set of tasks which will be periodically dispatched. This period is called a major frame and is partitioned into 3.2-ms subframes. Tasks are statically allocated to these subframe slots. Therefore, all task execution times must be less than or equal to 3.2 ms. If an application requires more time, it must be decomposed into a sequence of 3.2-ms tasks. In Version R and Version V, the operating system was generalized to allow a task to utilize any number of 1.6-ms intervals. In all versions, the application designer must allocate the tasks in a preplanned schedule table.

Figure 4 shows a typical task assignment to the subframes for a six-processor configuration. In this schedule, the application designer has defined a major frame containing 32 subframes.

The tasks are named by three-character identifiers (e.g., IC1, IC2, MLT, LAT). The application tasks are scheduled three times in the major frame. This is referred to as a "triple-frame" schedule. All the executive tasks except the Interactive Consistency Tasks are scheduled once per major frame. In a "single-frame" schedule, the application tasks are only scheduled once in the table. In such a schedule, the frame is shorter, but the operating system overhead is proportionately larger, as seen subsequently.

Since the SIFT system is reconfigurable, there must also be schedules for five-, four-, three-, and two-processor configurations (not shown). Although the Local Executive executable code and schedule table are identical on every processor, each executive uses a different section of the schedule table. Each section of the schedule table is identified by the ordered pair (NW,VPN), where the NW field indicates the number of working processors in the configuration, and the VPN field is the number of the virtual processor which uses this section of the schedule. Every physical processor has a virtual processor number assigned to it. After a reconfiguration, this virtual processor number may change. Since any processor may fail, the new virtual processor number cannot be predetermined. Thus, each processor contains all the schedule table sections. The schedule table then consists of a sequence of these sections which are initialized in BDX-930 assembly code.

Task table. The task table contains information specific to each task in the system. The following

SUBFRAME	PROCESSOR 1	2	3	4	5	6
0	IC1	IC1	IC1	---	---	---
1	IC2	IC2	IC2	IC2	---	---
2	IC3	IC3	IC3	IC3	IC3	IC3
3	MLT	---	MLT	MLT	MLT	MLT
4	---	GUT	GUT	GUT	GUT	GUT
5	PIT	PIT	---	PIT	PIT	PIT
6	LAT	---	LAT	LAT	LAT	LAT
7	---	---	---	---	---	---
8	---	---	---	---	---	---
9	IC1	IC1	IC1	---	---	---
10	IC2	IC2	IC2	IC2	---	---
11	IC3	IC3	IC3	IC3	IC3	IC3
12	MLT	---	MLT	MLT	MLT	MLT
13	---	GUT	GUT	GUT	GUT	GUT
14	PIT	PIT	---	PIT	PIT	PIT
15	LAT	---	LAT	LAT	LAT	LAT
16	---	---	---	---	---	---
17	---	---	---	---	---	---
18	IC1	IC1	IC1	---	---	---
19	IC2	IC2	IC2	IC2	---	---
20	IC3	IC3	IC3	IC3	IC3	IC3
21	MLT	---	MLT	MLT	MLT	MLT
22	---	GUT	GUT	GUT	GUT	GUT
23	PIT	PIT	---	PIT	PIT	PIT
24	LAT	---	LAT	LAT	LAT	LAT
25	---	---	---	---	---	---
26	---	---	---	---	---	---
27	ERT	ERT	ERT	ERT	ERT	ERT
28	FIT	FIT	FIT	FIT	---	FIT
29	RET	RET	RET	RET	RET	RET
30	---	---	---	---	---	---
31	CLT	CLT	CLT	CLT	CLT	CLT

where

IC1, IC2, IC3 = Interactive Consistency Tasks
MLT, GUT, PIT, LAT = Application Tasks
ERT = Error Task
FIT = Fault Isolation Task
RET = Reconfiguration Task
CLT = Clock Task

Figure 4. Typical task assignment in SIFT.

Pascal record defines its structure:

```
TT: ARRAY[TASK] OF RECORD
    CAUSE: (TASKTERM,CLOCKINT,SYSTEMSTART);
    BUFS: INTEGER;
    ERRORS: INTEGER;
    STKPTR: INTEGER;
    STATE: ARRAY[0..128] OF INTEGER;
    END;
```

Most of these fields are initialized and managed by the operating system. Only the BUFS field and the initial state of the task must be initialized by the applications programmer. The BUFS field points to a list of the buffer numbers in the Buffer information array (described below). This list defines the output variables of the task. The initial state holds the task starting location, terminating routine location, and initial register values. Other fields in the task table record are used by the scheduler. They contain the following information:

CAUSE — the reason for entry into the scheduler

ERRORS — the number of times the task failed to complete

STATE — the state of the task (including registers, restart address, and stack area) upon interrupt

STKPTR — the value of the stack pointer register upon task termination or clock interrupt

Buffer information array. Before the Buffer information (BINF) array can be constructed, the application designer must enter in BDX-930 assembly code a list of EQU instructions identifying each buffer "name" with a buffer "number"

```
ERRER  EQU 33
GEREC  EQU 34
       ⋮
```

These buffer names are simply convenient synonyms for the buffer numbers. The BINF array is essentially a memory pool containing lists of buffer names which are pointed to by the BUFS field of the task table. Each list is terminated by a zero field. The details of the assembly code which initializes this structure are not important for this discussion. This list is used in the Reconfiguration Task to rebuild the BT array, described next.

Buffer table. The buffer table, BT, is used by the Local Executive and generated by the Global Executive. The BT array is the central data structure used by the system for redundancy management. This structure maps the logical buffer names to physical datafile locations. Since SIFT uses replicated tasks to achieve fault tolerance, each data value (i.e., buffer) is calculated by several tasks and is broadcast to specific locations in the datafile of each processor. The BT array maintains the datafile locations of all the replicates of the data.

The BT array, shown below, is essentially a function mapping a buffer number into a vector indicating where its replicated values reside.

```
BT: ARRAY[0..MAXBUFS] OF RECORD
    DBX: INTEGER;
    AD: ARRAY[0..MAXPROCESSOR] OF INTEGER;
    END;
```

The datafile offset (DBX) must be entered by the application designer. This offset describes the location of the data within the 128-word mailbox. For example, buffer number 10 might have an offset of 8. Two different buffer numbers may be assigned the

same datafile offset (DBX), but the application designer must be careful not to utilize both at the same time—i.e., they must be time multiplexed. The ability to time multiplex datafile locations was sacrificed in Version R to enable an efficient reconfiguration process. The AD array is constructed by the Reconfiguration Task from information in the task schedule, task table, and BINF array and from data from the Fault Isolation Task. If processor i computes the buffer, then AD[i] contains the location of processor i output in the datafile. If processor i does not compute the buffer, then AD[i] = −1. For example, the BT array entry relating to the processor 2 datafile is shown in figure 5. Buffer item 10 is found at an offset of 8 into the processor mailbox and was produced by processors 0, 2, 3, and 5.

DBX:	8
AD[0]:	8
AD[1]:	-1
AD[2]:	904
AD[3]:	264
AD[4]:	-1
AD[5]:	520
AD[6]:	-1
AD[7]:	-1

Figure 5. Detailed description of BT[10].

Vote schedule and POSTVOTE array. The vote schedule is constructed in parallel with the task schedule. The vote schedule in figure 6 corresponds to the task schedule of figure 4. Unlike the task schedule, there is only one vote schedule for each level of configuration (i.e., all processors in the configuration use the same vote schedule). The schedule contains a list of items to be voted before the task scheduled for that subframe is executed. The result of this vote is placed in the POSTVOTE array. The restriction of allowing only one vote schedule for each configuration level guarantees that all good processors contain exactly the same data in the POSTVOTE buffers—even if their schedules do not execute tasks which use all the data. Although at first this appears wasteful, it simplifies the reconfiguration process. Since all data are available on every processor during reconfiguration, it is not necessary

```
SUBFRAME        BUFFERS TO BE VOTED
--------        -------------------

   0
   1            EXPEC
   2            NDR
   3            LOCK  XRESE
   4            QX    QY    QZ
   5            PSIN  PHIN  RN    QDELY QLATM TIMER
   6            CMDEL QDELZ CMDTH QPITM
   7            CMDAI CMDRU
   8
   9
  10            EXPEC
  11            NDR
  12            LOCK  XRESE
  13            QX    QY    QZ
  14            PSIN  PHIN  RN    QDELY QLATM TIMER
  15            CMDEL QDELZ CMDTH QPITM
  16            CMDAI CMDRU
  17
  18
  19            EXPEC
  20            NDR
  21            LOCK  XRESE
  22            QX    QY    QZ
  23            PSIN  PHIN  RN    QDELY QLATM TIMER
  24            CMDEL QDELZ CMDTH QPITM
  25            CMDAI CMDRU
  26
  27
  28
  29            GEREC GEMEM
  30
  31
```

Figure 6. Typical SIFT vote schedule.

to transfer data to a processor when its new schedule contains a task it previously had not executed.

Global Executive

The Global Executive performs four major functions:

1. Clock synchronization
2. Error report analysis and identification of faulty processors
3. Logical removal of a faulty processor via reconfiguration
4. Interactive consistency

These functions are performed by a set of tasks—the Clock Synchronization Task, the Error Task, the Fault Isolation Task, the Reconfiguration Task, and the Interactive Consistency Task. The details of the synchronization process are not presented here.

When a processor fails, the immediate effect of its errors is masked by the vote function. The voter records the number of errors produced by each processor. Once during each major frame, the Error Task condenses the local error data and broadcasts the information. All processors now have a systemwide record of the errors produced during the

```
VAR
   ERAD AT 16#4000: ARRAY[1..6,1..MAXSUBFRAME,1..8] OF INTEGER;
   ERADPT: INTEGER;

GLOBAL FUNCTION SCHEDULER(CAUSE:SCHED_CALL; STATE:INTEGER):INTEGER;
VAR  T1,I:INTEGER;
BEGIN
   TSKFN := CLOCK;                              (* - PERFORMANCE MONITOR - *)
   TT[TASKID].STKPTR := STATE;
   IF CAUSE<>TASKTERMINATION THEN   (* --- CLOCK INTERRUPT --- *)
      BEGIN

      IF (TASKID<>NULLT) THEN       (* TASK DID NOT COMPLETE *)
         BEGIN TT[TASKID].ERRORS := TT[TASKID].ERRORS + 1;
         BUILDTASK(TASKID);
         END
      ELSE TT[TASKID].STATUS := CLOCKINTERRUPT;

      IF SFCOUNT >= MAXSUBFRAME THEN (* START NEW MAJOR FRAME *)
         BEGIN SFCOUNT := 0;
         IF FRAMECOUNT >= MAXFRAME THEN FRAMECOUNT := 0
         ELSE FRAMECOUNT := FRAMECOUNT+1; GFRAME := GFRAME+1;
         IF ERADPT = 6 THEN ERADPT := 1       (* - PERFORMANCE MONITOR - *)
         ELSE ERADPT := ERADPT + 1;           (* - PERFORMANCE MONITOR - *)
         END
      ELSE SFCOUNT := SFCOUNT+1;

      TSCHEDULE;                    (* SELECT NEW TASK *)

      BCLOCK := CLOCK;                        (* - PERFORMANCE MONITOR - *)
      VSCHEDULE;                    (* PERFORM VOTE *)
      BCLOCK := CLOCK - BCLOCK;               (* - PERFORMANCE MONITOR - *)

      ERAD[ERADPT,SFCOUNT,1] := GFRAME;       (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,2] := SFCOUNT;      (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,3] := TSKFN;        (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,4] := BCLOCK;       (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,5] := RCLOCK;       (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,6] := XCLOCK;       (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,7] := 0;            (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,8] := 0;            (* - PERFORMANCE MONITOR - *)
      END
   ELSE                                (* --- TASK TERMINATION --- *)
      BEGIN T1 := TSKFN - TSKST;               (* - PERFORMANCE MONITOR - *)
      IF T1 > TTIME[TASKID] THEN               (* - PERFORMANCE MONITOR - *)
         TTIME[TASKID] := T1;                  (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,7] := TSKST;         (* - PERFORMANCE MONITOR - *)
      ERAD[ERADPT,SFCOUNT,8] := TTIME[TASKID]; (* - PERFORMANCE MONITOR - *)
      TASKID := NULLT;
      END;
   SCHEDULER := TT[TASKID].STKPTR;
   TSKST := CLOCK;                             (* - PERFORMANCE MONITOR - *)
END; (* SCHEDULER *)
```

Figure 7. SCHEDULER procedure with instrumentation.

past frame. The Fault Isolation Task searches the error data to locate faulty processors. The Fault Isolation Task then broadcasts a value indicating which processors are faulty. This value is voted and used by the Reconfiguration Task to compute the set of "good" processors. The Reconfiguration Task does not physically remove a faulty processor from the configuration. The good processors merely agree to ignore outputs from the faulty processor. Thus, reconfiguration may be accomplished by changes to the internal data structures.

The Reconfiguration Task basically selects a new schedule table and regenerates the buffer table (described previously) to accomplish the logical reconfiguration. Since the algorithm used to determine which processor must be eliminated is decentralized, it is essential that the algorithm is designed so every good processor makes exactly the same decision at exactly the same time. This must be done correctly even in the presence of a malicious processor (i.e., one that sends good data to some processors and erroneous data to others). This is accomplished by use of the "interactive consistency algorithm" developed by SRI International (ref. 4) and discussed in the section "Interactive Consistency overhead."

SIFT Scheduler

The scheduler consists of two major components—the assembly code interrupt handler and the Pascal procedure SCHEDULER. The Pascal procedure is called from assembly code whenever any one of the following three events occurs: (1) system startup, (2) task termination, or (3) clock interrupt. The SCHEDULER has two primary responsibilities after a clock interrupt—vote the data scheduled for the subframe and dispatch the next task according to the information in the schedule table.

Under Version B of the SIFT Operating System, an application designer has to divide a process that takes longer than 3.2 ms into a series of 3.2-ms tasks. An entry must be made in the schedule table for each subframe in which the process would run. To spare the designer the work of partitioning the process and to reduce the size of the schedule table, the structure of the schedule table was changed to allow the designer to define how many 1.6-ms interrupts the task should use.

Subsystem Overhead Measurement

Instrumentation of Operating System

Since the SCHEDULER controls voting and the dispatching of tasks (and therefore the Global Executive), the measurement instrumentation was added to the SCHEDULER. The SCHEDULER with the measurement code is shown in figure 7. An array, ERAD, was added to store the data during a test. ERAD is a three-dimensional array with indices ERADPT, SFCOUNT, and DVI, which differentiate the 6 major frames, 32 subframes, and 8 data items, respectively. To retrieve the performance data, the SIFT processors were allowed to run an arbitrary amount of time (5 to 10 s) and then were halted manually from the host processor. The portion of the processors' memories containing the ERAD array was copied to a disk file and analyzed by offline programs. The eight data items are as follows:

GFRAME	the major frame count (nonrepeating count)
SFCOUNT	the subframe count (0..MAXSUBFRAME)
TSKFN	the time the previous task finished
BCLOCK	the vote time for the subframe
RCLOCK	not used
XCLOCK	not used
TSKST	the time at which the task for the subframe started
TIME	the maximum task execution time

Figure 8 shows the components of a subframe. All the variance in execution time results from either the voter or the task itself. The time required for dispatching a task was calculated by subtracting the vote time from the total time spent in the SCHEDULER routine. This overhead then includes the time needed for the instrumentation. Figure 8 shows the task schedule overhead to be nominally 270 μs. An analysis of the voter and the Global Executive tasks follows.

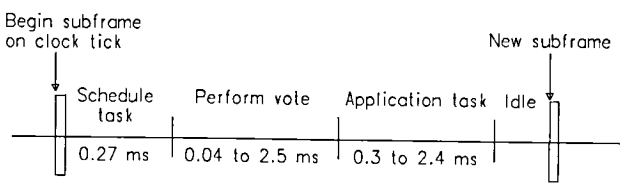

Figure 8. Components of a subframe.

Vote Overhead

Voting is performed at the beginning of every subframe prior to the execution of the application task. The operating system scans the vote schedule

table to determine which data buffers are to be voted. For each such data buffer, the VOTE routine is called. The VOTE routine uses the BT array to retrieve the replicated versions of the data from the datafile. After the data are retrieved, either VOTE3 or VOTE5 is called for three-way voting or five-way voting, respectively, depending on the number of values found. Unfortunately, the time required for voting is affected by the following four factors:

1. *VS*, the type of vote—three-way or five-way (determined by the number of task replicates which generate data)
2. *NV*, the number of data buffers to be voted, as indicated in the vote schedule
3. *HP*, the position of the data-producing tasks in the schedule table
4. *NW*, the number of working processors in the configuration

The characteristics of the vote time are different for each of the operating system versions. These vote times will be referred to as V_B, V_R, and V_V. Thus, the vote time for each of these versions is effectively a function defined on a four-dimensional space as follows:

$$V_B(VS, NV, HP, NW)$$
$$V_R(VS, NV, HP, NW)$$
$$V_V(VS, NV, HP, NW)$$

Obviously, it is difficult to clearly present the details of an empirical function defined on a four-dimensional space. In this paper, this is attempted by illustrating different planes in the four-dimensional space.

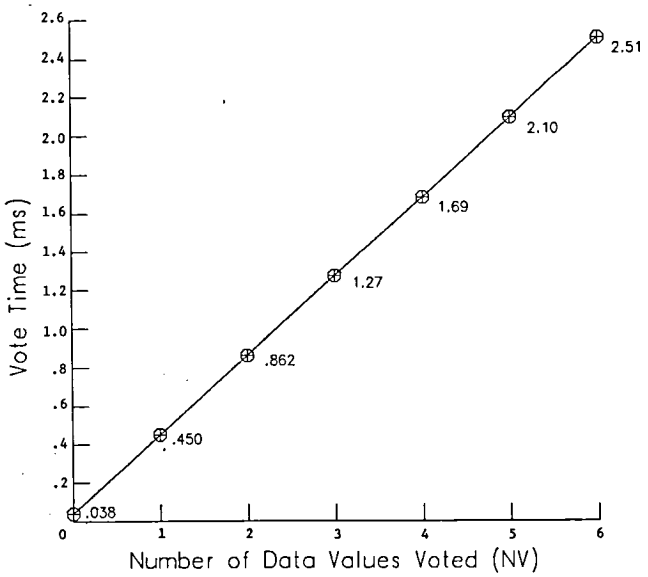

Figure 9. Five-way vote times $V_B(5, NV, 6, 6)$.

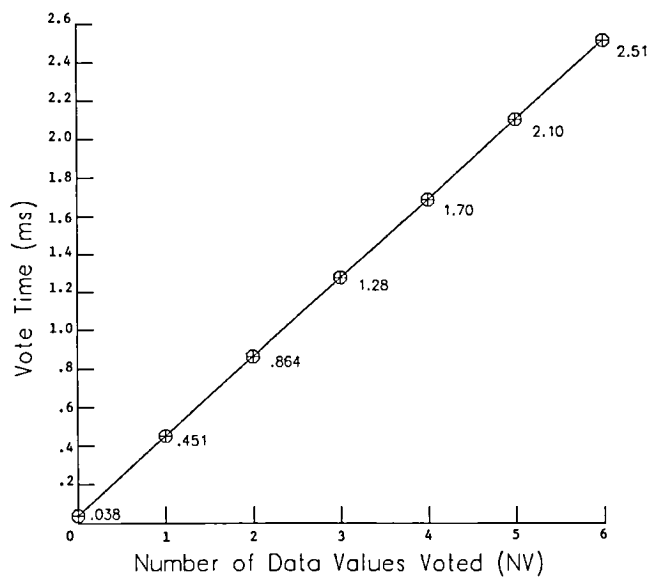

Figure 10. Three-way vote times for $V_B(3, NV, x, 6)$, where $x \in \{3, 4, ..., 8\}$.

Version B vote times. First, looking only at a six-processor configuration ($NW = 6$) and assigning the highest task replicate to processor 6 ($HP = 6$), we obtain the graph in figure 9 for $V_B(5, NV, 6, 6)$. Surprisingly, the vote time for the three-way vote, $V_B(3, NV, 6, 6)$ is virtually identical to that of the five-way vote. A single five-way vote requires 0.412 ms, whereas a single three-way vote requires 0.413 ms. (See fig. 10.) Although the VOTE3 routine does execute faster, the code which retrieves data from the datafile (see fig. 11) continues until either all eight fields of the BT array are examined or five good values (i.e., not -1) are found.

Thus, when there are only three data-producing tasks, the loop does not terminate until "I>MAXPROCESSOR." (MAXPROCESSOR is 7.) This additional looping time is almost exactly equal to the difference between the VOTE3 and VOTE5 execution times. If five-way voting is not required on any tasks, then the loop test can be changed to "UNTIL (J=3) OR (I>MAXPROCESSOR)." With this modification, the system will be referred to as "TRIAD SIFT." The three-way vote in TRIAD SIFT will be indicated by appending an * after the subscript letter denoting the version (e.g., V_{B*}). The $V_{B*}(3,NV,6,6)$ vote times for the TRIAD SIFT are compared with

```
VAR BT: ARRAY[0..MAXBUFFERS] OF RECORD
                               DBX: INTEGER;
                               AD: ARRAY[0..MAXPROCESSOR] OF INTEGER;
                               END;

PROCEDURE VOTE(B: BUFFER; DEFAULT: INTEGER);

VAR I,J,K: INTEGER;

BEGIN

J := 0; I := 0;
REPEAT
   K := BT[B].AD[I];                (* DATAFILE ADDRESS OF BUFFER B *)
   IF K >= 0 THEN
      BEGIN
      J := J + 1;
      P[J] := I;                    (* SAVE PROCESSOR NUMBER *)
      V[J] := DATAFILE[K];          (* RETRIEVE DATA FROM DATAFILE *)
      END;
   I := I + 1;
UNTIL (J=5) OR (I>MAXPROCESSOR);    (* UNTIL 5 VALUES FOUND OR SEARCH DONE *)

   .
   .     (* CALL APPROPRIATE VOTE FUNCTION; i.e. 5-WAY OR 3-WAY *)
   .

END;
```

Figure 11. Vote procedure in Version B.

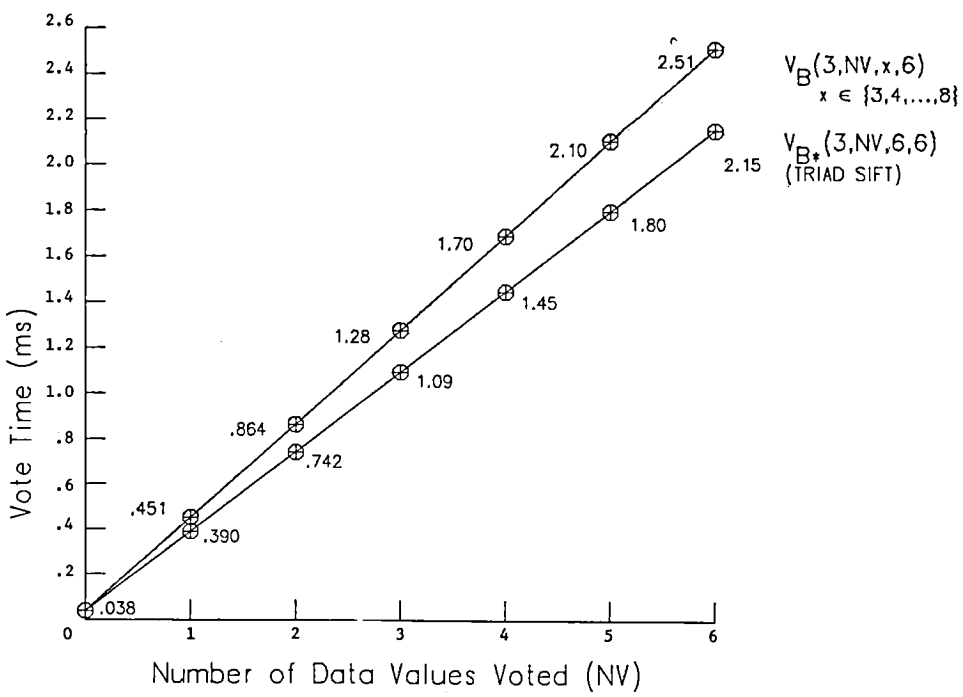

Figure 12. Vote times for three-way SIFT and TRIAD SIFT.

the normal SIFT vote times in figure 12. From this graph, the advantage of the above software modification when only three-fold redundancy is needed is clearly seen.

Because the vote time is a linear function of NV, the following formula is valid:

$$V_B(VS, NV, HP, NW) = C_B(VS, HP, NW) \cdot NV + B$$

where $C_B(VS, HP, NW)$ is the slope of the line and B is the y-intercept. The notation C_B essentially represents the vote time per buffer and will be referred to as the vote cost. The symbol B represents the basic overhead when no buffers are voted. It is independent of the other parameters of V_B and always has the value 0.038 ms.

Next, the impact of the position of the tasks in the schedule table on the vote time is illustrated. In a six-processor configuration, there are six ways to assign the five task replicates to the processors. However, the only factor which influences the vote time is the highest numbered processor which is running the task; i.e., in all the following five assignments of task $t1$, processor 6 is the highest processor executing the task, and thus all five assignments have the same vote time.

Processor number					
1	2	3	4	5	6
	$t1$	$t1$	$t1$	$t1$	$t1$
$t1$		$t1$	$t1$	$t1$	$t1$
$t1$	$t1$		$t1$	$t1$	$t1$
$t1$	$t1$	$t1$		$t1$	$t1$
$t1$	$t1$	$t1$	$t1$		$t1$

For each of the above task assignments, $HP = 6$. For the remaining assignment

Processor number					
1	2	3	4	5	6
$t1$	$t1$	$t1$	$t1$	$t1$	

the highest processor is 5; thus, $HP = 5$. The vote costs $C_B(5, HP, 6), C_B(3, HP, 6)$, and $C_{B*}(3, HP, 6)$ are given as a function of HP in figure 13. (Note that HP is the physical processor number, which is derived from the slot position on the broadcast bus. Therefore, even if the number of working processors is less than eight, HP can be made equal to 8 by placing one of the good processors in slot 8.) The vote times in Version B are independent of NW. As can be seen in figure 13, the vote costs can be dependent on HP and therefore on the task schedule constructed by the application designer. This is not a desirable attribute, since a task which runs within its time limit in one schedule may not have enough time to run if the schedule is slightly modified.

It is noteworthy that the HP dependency arises from the (J=5) test of the data retrieval loop. By simply removing this test from the Boolean expression, this dependency can be eliminated. The vote time would then always be worst case (i.e., $C_B(5, \text{MAXPROCESSOR}, 6)$), but the verification process would be simpler.

Version R vote times. In Version R, a minor modification was made to the data retrieval loop code. (See fig. 14.) This modification reduced the vote time in certain cases but introduced the additional complexity that the vote time depends on NW. The data retrieval loop in Version R differs from that in Version B in two significant aspects. First, the loop termination logic now refers to NW rather than MAXPROCESSOR. Second, the variable I refers to virtual processor number rather than physical processor number as in Version B. This is a consequence of the new design. The BT array in Version R is indexed by the number of working processors, NW, and buffer number, B. (See the section on reconfiguration overhead.) The BT array in Version B was indexed by buffer number alone. Thus, whereas the Version B vote times were influenced by the highest physical-processor-producing data, the Version R vote times are influenced by the highest virtual processor number. Thus in this section, HP will become HVP, which denotes the highest virtual processor which produces the data being voted. In figure 15, the vote costs $C_R(3, 3, NW)$, $C_R(5, 5, NW)$, and $C_R(5, 6, 6)$ are given as a function of the number of working processors, NW. The value of $C_R(5, 6, 6)$ is greater than that of $C_R(5, 5, 6)$ because an extra BT array access is needed. The three-way vote cost dependency on NW occurs because the BT array search loop terminates when I becomes greater than NW. In figure 16, the effect of HVP is illustrated. The three-way vote cost $C_R(3, HVP, NW)$ is seen to be independent of HVP. However, as can be seen in figure 17, the TRIAD SIFT three-way vote cost is dependent on the parameter HVP. In figure 18, TRIAD SIFT is compared with Version R SIFT.

The NW dependency of this version arises from the (I>NW) test in the REPEAT-UNTIL loop. By returning this to the (I>MAXPROCESSOR) test, the NW dependency can be eliminated. Although the (I>NW) test results in a reduced vote time for smaller SIFT configurations (e.g., after several reconfigurations), additional vote overhead complexity

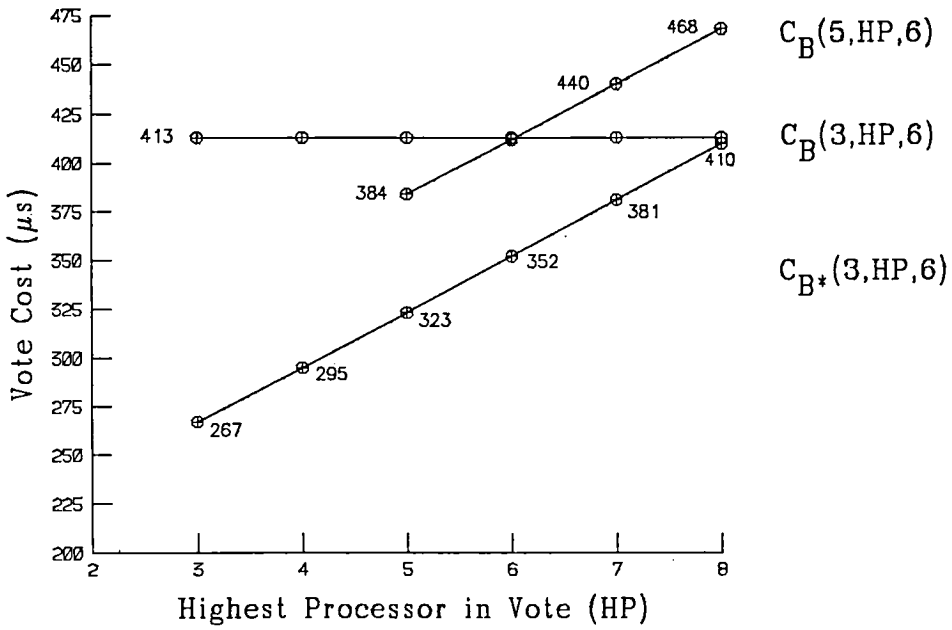

Figure 13. Vote costs for $C_B(5, HP, 6)$, $C_B(3, HP, 6)$, $C_{B*}(3, HP, 6)$.

```
VAR BT: ARRAY[0..MAXPROCESSOR,0..MAXBUFFERS] OF INTEGER;

PROCEDURE VOTE(B: BUFFER; DEFAULT: INTEGER);

VAR I,J,K: INTEGER;

BEGIN

J := 0; I := 0;
K := BT[NW,B];        (* BIT VECTOR OF DATA PRODUCING PROCESSORS *)
REPEAT
   IF ODD(K) THEN     (*  PROCESSOR I COMPUTES BUFFER *)
      BEGIN
      J := J + 1;
      P[J] := I;                          (* SAVE PROCESSOR NUMBER *)
      V[J] := DATAFILE[VTODF[I]+B];       (* RETRIEVE DATA FROM DATAFILE *)
      END;
   K := K DIV 2;                          (* SHIFT NEXT PROCESSOR BIT TO LSB *)
   I := I + 1;
UNTIL (J=5) OR (I>NW);       (* UNTIL 5 VALUES FOUND OR SEARCH DONE *)

   .
   .     (* CALL APPROPRIATE VOTE FUNCTION; i.e. 5-WAY OR 3-WAY *)
   .

END;
```

Figure 14. Vote procedure in Version R.

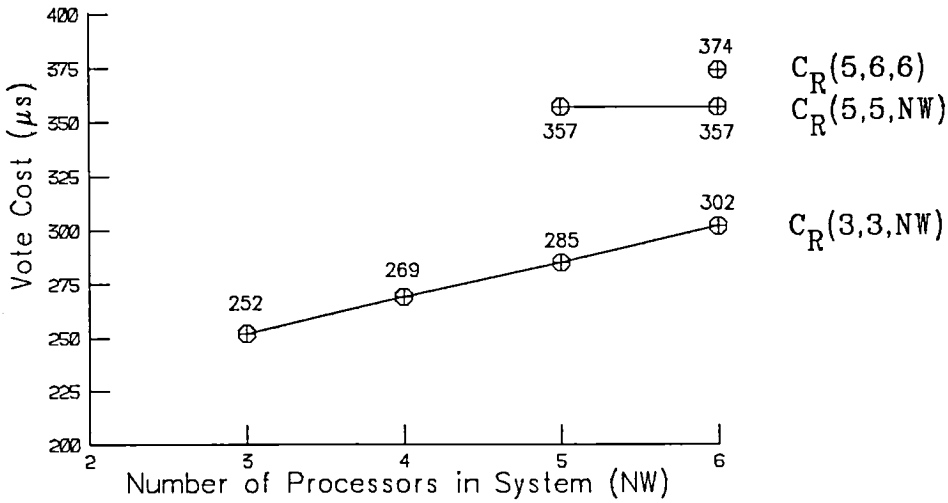

Figure 15. Vote costs of optimized system.

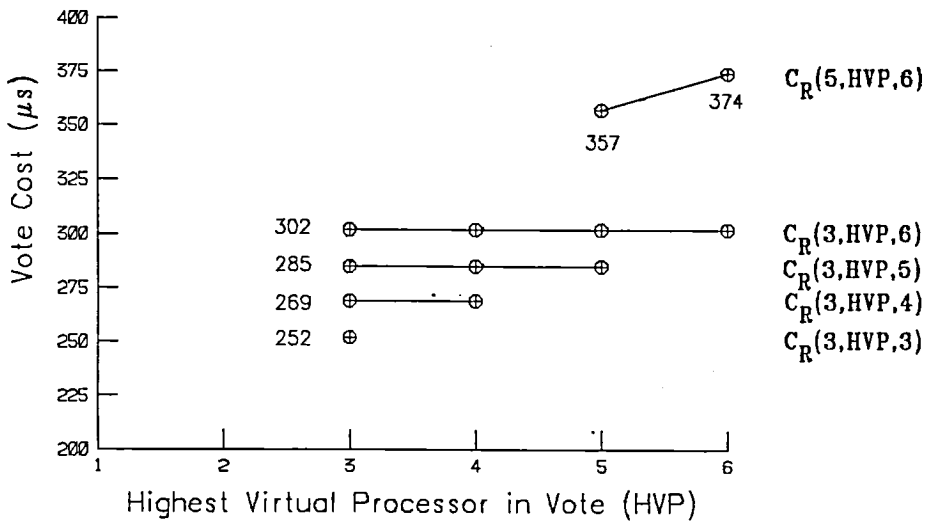

Figure 16. Vote costs for $C_R(5, HVP, 6)$ and $C_R(3, HVP, NW)$.

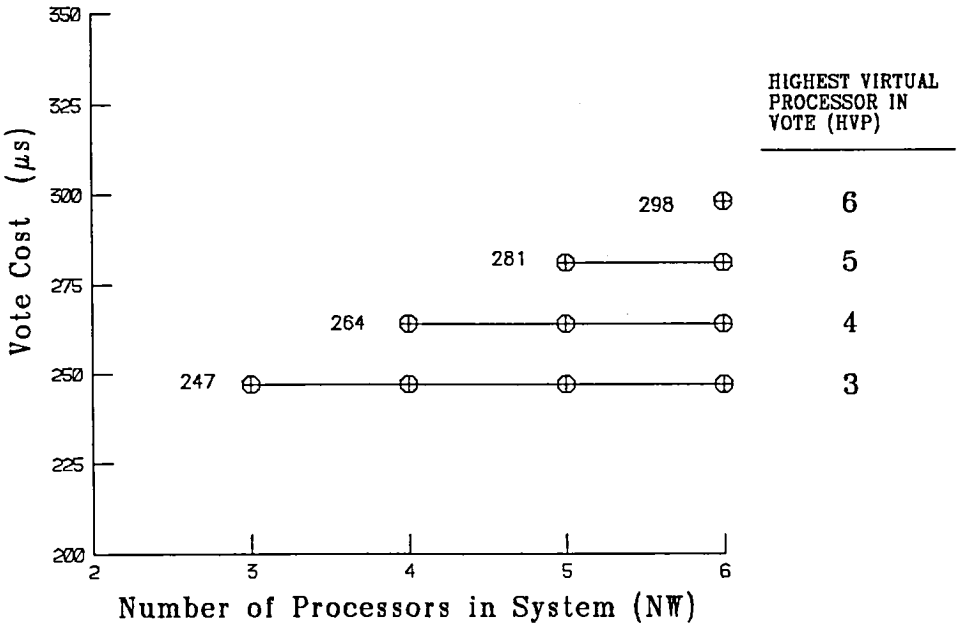

Figure 17. Vote costs for $C_{R*}(3, HVP, NW)$ of TRIAD SIFT.

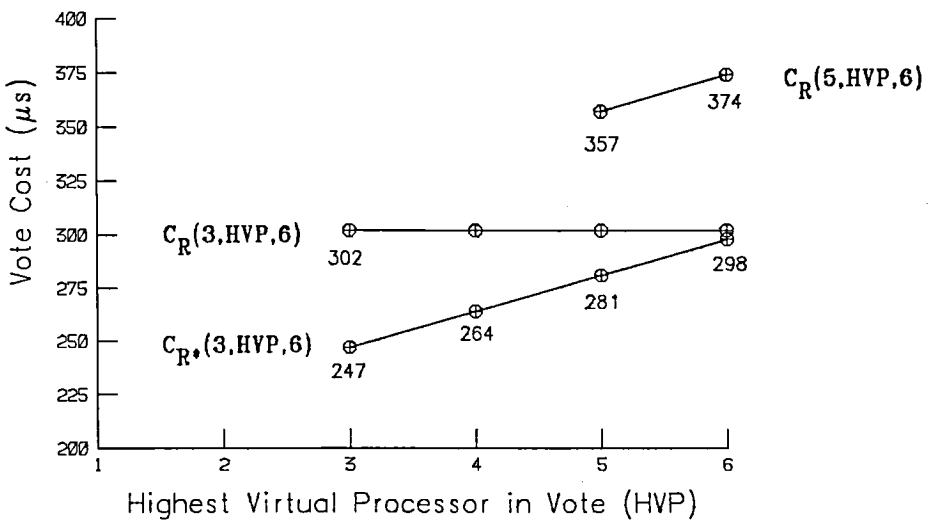

Figure 18. Vote costs for $C_R(5, HVP, 6)$, $C_R(3, HVP, 6)$, $C_{R*}(3, HVP, 6)$.

```
VAR BT: ARRAY[0..MAXPROCESSOR,0..TASKS] OF INTEGER;

PROCEDURE VOTE(TK: TASK; DEFAULT: INTEGER);

VAR I,J,K: INTEGER;
    B: BUFFER;

BEGIN

J := 0;  I := 0;
K := BT[NW,TK]                (* BIT VECTOR OF DATA-PRODUCING PROCESSORS *)
REPEAT
   IF ODD(K) THEN             (* PROCESSOR I EXECUTED TASK TK *)
      BEGIN
      J := J + 1;
      P[J] := I;              (* SAVE VIRTUAL PROCESSOR NUMBER *)
      DF[J] := VTODF[I];      (* SAVE DATAFILE OFFSET *)
      END;
   K := K DIV 2;              (* SHIFT NEXT PROCESSOR BIT TO LSB *)
   I := I + 1;
UNTIL (J=5) OR (I>NW);

I := TT[TK].BUFS;       (* RETRIEVE INDEX OF FIRST BUFFER OF TASK *)
B := BINF[I];           (* RETRIEVE BUFFER NUMBER *)
WHILE B > 0 DO
   BEGIN
   .
   .     (* CALL APPROPRIATE VOTE PROCEDURE, i.e. 5-WAY OR 3-WAY *)
   .
   I := I + 1;
   B := BINF[I]         (* RETRIEVE NEXT BUFFER NUMBER *)
   END;
```

Figure 19. Vote procedure in Version V.

is also obtained. This overhead complexity significantly increases the effort to validate the system, since one must insure that adequate time is present for all the tasks to run under all possible configurations and all possible schedules that the system may encounter. Perhaps the workload requirements of the degraded configurations will require this reduced overhead (i.e., for small values of NW), since processing power is scarce, but a serious price is paid for this during the validation effort.

Version V vote times. In Version V, the explicit purpose of the operating system modification was to decrease vote costs. An analysis of the VOTE procedure showed that a great deal of time was spent indexing into the BT array for each buffer. After the modifications of Version R, the information in the BT array was reduced to a bit vector representing the set of processors which produces the buffer. Since the set of processors that runs a task which computes a buffer is equivalent to the set of processors that produces the buffer, the approach taken was to modify the BT array so the VOTE procedure could manipulate a task instead of a buffer. (See fig. 19.)

The BT array was modified to be indexed by task rather than by buffer number, and the vote schedule was changed to a list of task names instead of buffer numbers. As seen in figure 20, although Version V pays an initial penalty and actually takes longer when voting one buffer, it shows significant improvement over Version R (and therefore also over Version B) for more than one buffer. In Versions B and R, the basic overhead time B is a constant, but in Version V, all the configuration dependency is in the basic overhead. Thus, the following formula describes the Version V vote overhead V_V:

$$V_V = C_V(VS) \cdot NV + B_V(VS, HVP, NW)$$

Since B_V is no longer a constant overhead, it will be referred to as vote bias. Figure 21 shows this vote bias relationship to VS, HVP, and NW. B_V is dependent on NW, as shown by the solid lines $B_V(5, HVP{=}NW, NW)$ and $B_V(3, x, NW)$. B_V is only dependent on HVP for $NW = 6$ in a five-way vote. For TRIAD SIFT, B_{V*} is independent of NW and only dependent on HVP, as shown by the dashed lines $B_{V*}(3, x, NW)$. The B_V term only contains the data retrieval overhead. The actual time spent voting is represented by C_V and is dependent only on the type of vote done. For a three-way vote, $C_V(3) = 0.079$ ms. A five-way vote has a cost, $C_V(5)$, of 0.107 ms.

The redesign of the vote system moved the dependencies on NW and HVP to the vote bias. However, as in the other versions, this dependency can be eliminated by removing the (I>NW) and (J=5) tests from the UNTIL loop.

TABLE I. ERROR IMPACT ON THREE-WAY VOTE

Faulty processor(s)	Increase in vote time, ms
1	0.039
2	.031
3	.031
1,2	.096

TABLE II. ERROR IMPACT ON FIVE-WAY VOTE

Faulty processor(s)	Increase in vote time, ms
1	0.040
2	.032
3	.032
4	.032
5	.032
1,2	.079
1,3	.072
1,4	.072
1,5	.079
2,3	.071
2,4	.063
2,5	.063
3,4	.063
3,5	.061
4,5	.061
1,2,3	.072
1,2,4	.064
1,2,5	.064
1,3,4	.064
1,3,5	.064
1,4,5	.072
2,3,4	.064
2,3,5	.064
2,4,5	.064
3,4,5	.064

Error impact on vote time. The vote time measurements given in the previous sections were made only when the data replicates were identical. Unfortunately, the vote time is increased if some of

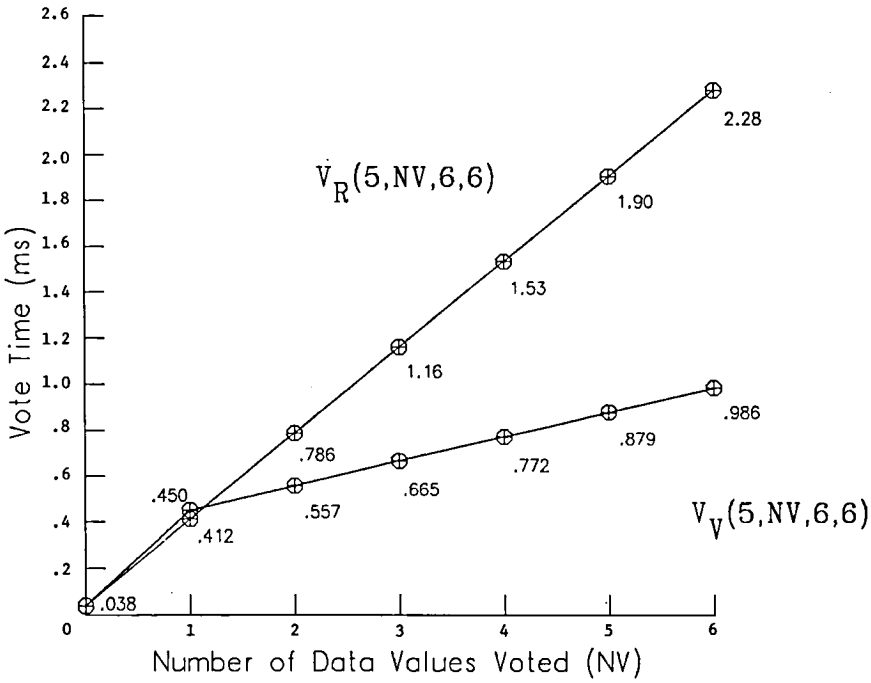

Figure 20. Comparison of V_V and V_R vote times.

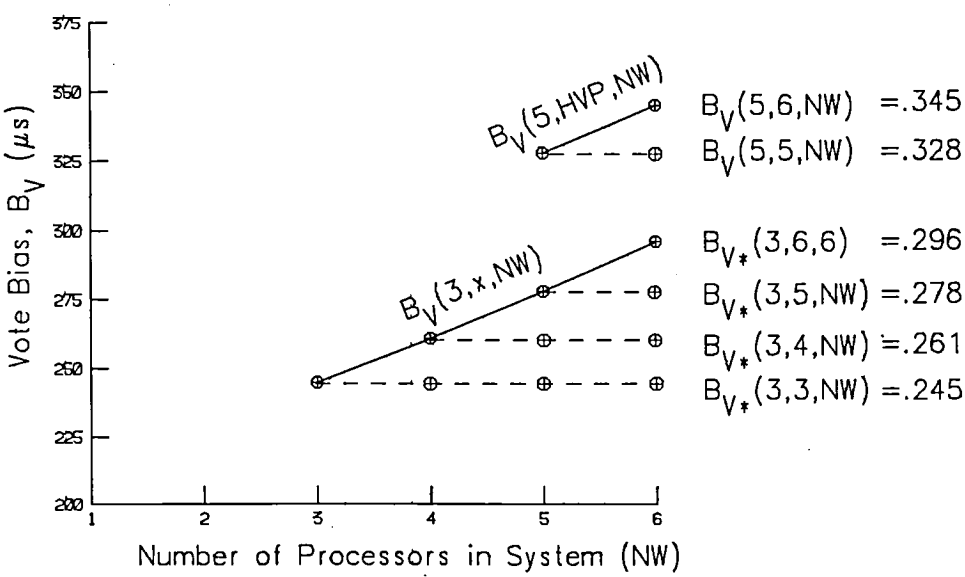

Figure 21. Vote bias dependency on *NW* and *HVP* in V_V.

the data values are erroneous. Furthermore, the increase is slightly dependent on the particular processor which generated the erroneous data because of the structure of the IF..THEN..ELSE statements in the VOTE procedure. The increases in vote time are given in table I and table II. From the tables it is obvious that the application designer must be very careful to insure that the worst-case vote time is accommodated when generating the vote and task schedule tables.

Executive Task Overhead

Reconfiguration overhead. To maintain a high level of reliability, the SIFT Global Executive removes faulty processors from the system, or reconfigures. The process is divided into three tasks. The Error Task transfers the local error data to the Global Executive via an error report. The Fault Isolation Task uses the error report from each processor to locate faulty processors. The Reconfiguration Task determines if a reconfiguration is necessary. During normal operation, each Global Executive task uses one 3.2-ms task slot. If a fault occurs and results in a reconfiguration, the Reconfiguration Task utilizes resources significantly in excess of 3.2 ms.

The exact time for reconfiguration depends on the number of working processors, but the worst case was found in Version B to be 35.19 ms or 11 subframes. (See fig. 22.) Since the scheduling is static, it must be based on worst-case performance, and hence 11 subframes must be dedicated to the Reconfiguration Task, even though the vast proportion of time they are not being utilized. However, in Version B, rather than dedicate 11 subframes, the real-time clock interrupt was disabled during the Reconfiguration Task. This allowed the task to take as much time as necessary, even though it was allocated to only one subframe. This was clearly unacceptable, since a serious disruption of output data would occur during the reconfiguration process. Therefore, the SIFT Reconfiguration Task was redesigned at the NASA Langley Research Center.

As stated above, the worst-case time for a reconfiguration is equivalent to eleven 3.2-ms subframes under Version B. Most of this time is spent reconstructing the BT array. The reconstruction is needed because the physical to virtual processor mapping changes after every reconfiguration. This mapping is necessary because the voter operates on physical processors, whereas the task schedules (and therefore, the processor data production information) are necessarily built in reference to virtual processors. The Version R design enables the Reconfiguration Task to execute in one 3.2-ms subframe.

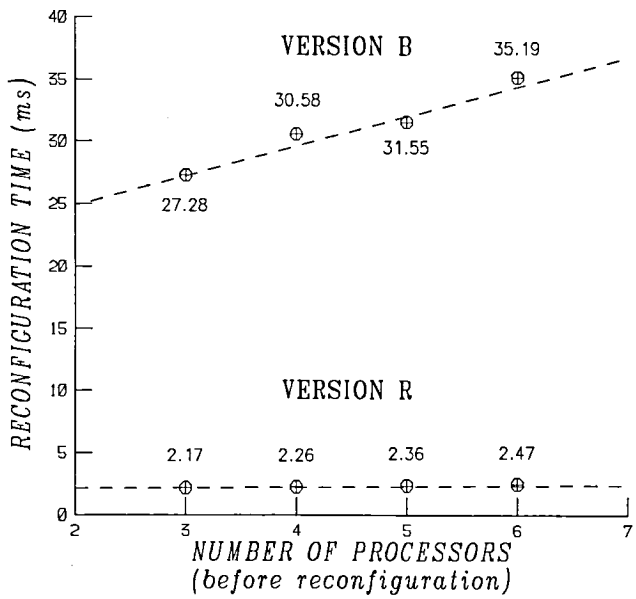

Figure 22. Reconfiguration times for Versions B and R.

The major points of the Version R redesign are

1. A buffer number defines the location of the buffer within the processor mailbox; i.e., each buffer item is assigned a unique mailbox address.
2. Array VTODF maps the virtual processor number to the corresponding physical processor mailbox offset in the datafile.
3. Arrays RTOV and VTOR map real to virtual processor numbers and vice versa, respectively.
4. The BT array is restructured to hold the virtual processor data production information for every configuration level.
5. The voter operates on virtual processors, and hence the ERROR array is represented in terms of virtual processors.
6. The Error Task translates the ERROR array information to physical processor numbers while constructing the error report.

A minor loss in generality comes about from point (1) above. Under Version B, two or more buffers could be assigned to the same mailbox location. This would be necessary if, for example, the system required more than 128 data buffers. This condition is not conceptually restrictive, since the datafile could be enlarged to 32K words (or more!) to allow 4000 buffers per processor. The BT array under Version R of the SIFT Operating System now has the form

```
BT : ARRAY[PROCESSOR, BUFFER] OF INTEGER;
```

For a configuration level of *NW* processors and buffer B, BT[NW,B] contains a bit map of the virtual processors that produce buffer B. Under Ver-

sion R, the BT array is filled during system initialization from task schedule information. The BT then contains valid information for all levels of system configuration. The Reconfiguration Task now builds arrays VTOR, RTOV, and VTODF. During a reconfiguration, since the BT array no longer needs to be rebuilt, the only chore is to select the proper task and vote schedules. The execution time for the Reconfiguration Task was reduced to about 2.5 ms for all levels of initial configuration.

Interactive Consistency overhead. Data values from the external environment are unreplicated and must be transferred to all computers of the system in a consistent manner; i.e., all computers must receive the same value. This could be accomplished by every processor reading the external source independently or by one processor reading the external source and then distributing the obtained value to the rest of the processors. In the first case, each processor might get a different value because of the inherent uncertainty of reading analog data. Hence, a subsequent exchange of the values read, along with a midvalue selection, is required to produce a value which is consistent across all processors. However, if one of the processors is malicious, i.e., sends different values to different processors, then the good processors can still end up with different values. (See fig. 23.) The second method will produce similar erroneous results if the single "input" processor is malicious. Note that although the good processors each decide on a slightly different value, they are both "good" in that the difference is only the slight difference in the redundant external sources. A midvalue selection on the replicated output channels would always result in a "good" output. However, if exact match voting is used to detect and isolate the fault, then serious problems can result, e.g., a good processor can be reconfigured out of the system. Thus, special "interactive consistency" algorithms are essential in fault-tolerant systems in which fault isolation and reconfiguration are performed. (See refs. 5 and 6.) In systems in which fault-masking is performed but no reconfiguration is attempted, such algorithms are unnecessary.

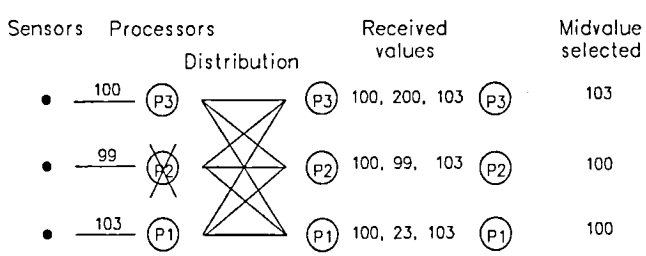

Figure 23. Distributing unreplicated data.

The overhead for these special interactive consistency algorithms can be very large. This overhead is especially severe because the failure modes that the algorithms eliminate may be rare events. In the absence of a thorough analysis demonstrating that the probability of these failure modes is negligible, interactive consistency algorithms must be used. In order to accommodate m faulty processors, the total number of processors, n, must be at least $3m + 1$. The number of messages required to obtain interactive consistency is on the order of n^{m+1}. Although five-way voting, which can deal with two internal faults, is supported in SIFT, the interactive consistency algorithm that was implemented can only handle one malicious fault. The simple flight control applications currently running in SIFT use 63 external sensor values, each of which goes through the interactive consistency algorithm. This requires 11.8 ms when no disagreements occur in the data. With faulty data present, the Interactive Consistency Tasks can utilize up to 13.4 ms.

The interactive consistency algorithm consists of the following steps:

1. The source value is input and distributed to the n processors.
2. The received values are exchanged m times.
3. A consistent value is obtained by use of a recursive algorithm. When $m = 1$, this reduces to determining a majority value.

The following execution times were measured for SIFT:

Step (1) 3.05 ms
Step (2) 2.22 ms
Step (3) 6.57 ms
Total 11.84 ms

Since the Interactive Consistency Tasks must be executed at the data sample rate, a large portion of the available CPU time is consumed, as shown in the table below.

Data sample period, ms	Utilization, percent
100	11.8
50	23.7
33	35.9
25	47.4

Execution times of executive tasks. The maximum execution times of the SIFT executive tasks were measured and are tabulated on page 20. The dispatch time represents the amount of time utilized

by the operating system prior to dispatching the executive task. This includes the vote time for this subframe. Therefore, the dispatch overhead is strongly dependent on the vote schedule. It should be noted that some variables voted during a particular subframe are not necessarily used by that task. The execution time column gives the time used by the executive task after being dispatched. The total time column is the sum of the dispatch time and execution time columns. A single frame is a major frame that contains one iteration of the sample application set.

A triple frame contains three iterations of the application set and therefore requires three iterations of the Interactive Consistency Tasks. The Global Executive Tasks are dispatched once every major frame. In particular, the Reconfiguration Task is executed at the major frame rate. Preliminary design studies recommended a major frame period of 100 ms in order to achieve the reliability requirements of SIFT.

Under Version B, the application set of four tasks utilizes seven 3.2-ms subframes. The executive overhead then is

Version B subsystem	Dispatch time, ms	Execution time, ms	Total time, ms	No. of 3.2-ms subframes
Interactive Consistency	1.7	11.8	13.5	5
Error Task	.3	.3	.6	1
Fault Isolation Task	.3	2.4	2.7	1
Clock Synchronization	.3	2.4	2.7	1
Reconfiguration Task	1.1	34.1	35.2	11
				19

19 subframes = 60.8 ms = 73.2% of an 83.2-ms single frame
29 subframes = 92.8 ms = 58.0% of a 160.0-ms triple frame

Under Version R, the application set of four tasks utilizes twelve 1.6-ms subframes. The executive overhead then is

Version R subsystem	Dispatch time, ms	Execution time, ms	Total time, ms	No. of 1.6-ms subframes
Interactive Consistency	1.4	11.8	13.2	10
Error Task	.9	.3	1.2	2
Fault Isolation Task	1.4	2.4	3.8	3
Clock Synchronization	.3	2.4	2.7	2
Reconfiguration Task	.9	2.4	3.3	3
				20

20 subframes = 32.0 ms = 62.5% of a 51.2-ms single frame
40 subframes = 64.0 ms = 52.6% of a 121.6-ms triple frame

Under Version V, the application set of four tasks utilizes ten 1.6-ms subframes. The executive overhead then is

Version V subsystem	Dispatch time, ms	Execution time, ms	Total time, ms	No. of 1.6-ms subframes
Interactive Consistency	1.3	11.8	13.1	10
Error Task	.7	.3	1.0	2
Fault Isolation Task	.7	2.4	3.1	2
Clock Synchronization	.3	2.4	2.7	2
Reconfiguration Task	.7	2.4	3.1	2
				18

18 subframes = 28.8 ms = 64.3% of a 44.8-ms single frame
38 subframes = 60.8 ms = 55.9% of a 108.8-ms triple frame

The overhead improvement in the subsequent versions of the operating system is readily seen in the decrease in the length of a triple frame. The decrease from Version B to Version R (i.e., 160.0 ms to 121.6 ms) is a result of the improved Reconfiguration Task. The further decrease from Version R to Version V (i.e., 121.6 ms to 108.8 ms) resulted from the decrease in vote time and the consequent ability to squeeze several tasks into one less subframe each. The higher percentage overhead of Version V results from the smaller major frame size and not from any increased inefficiency.

The impact of the fault tolerance mechanisms on performance can be seen by comparison with an equivalent simplex system. The overhead of such a simplex system is easily calculated from the available data. Without voting, the dispatch overhead would be about 270 μs, or less than 10 percent of a 3.2-ms subframe. The time needed to execute the four sample application tasks would be approximately four subframes, or 12.8 ms. A communications task, the equivalent of Interactive Consistency Task 1 (IC1), would still be needed and would require at most one 3.2-ms subframe. The executive task overhead would then be 20 percent of a small 16.0-ms major frame or 3.2 percent of a larger 100-ms major frame.

Discussion of Results

The SIFT operating system utilizes significant CPU resources to achieve fault tolerance in software. This overhead falls in two main categories:

1. The time required to vote the intertask communication variables at the beginning of each subframe
2. The time utilized by the executive tasks, especially the Interactive Consistency Tasks

The overhead from the first category, vote overhead, was found to be a linear function of the amount of data to be voted. Unfortunately, for as few as six data buffers, the vote overhead was in excess of 30 percent of a 3.2-ms subframe. The vote times were measured for three versions of the operating system. Also, a slight modification to the VOTE routine was discovered which enables a more efficient three-way vote if the system is run with only three-way voting, i.e., no five-way replicated tasks. This modification is referred to as TRIAD SIFT. The following table gives a basic comparison of the vote times (best values) for all versions of SIFT investigated with a six-processor configuration:

Version	Three-way vote time per buffer, ms	Five-way vote time per buffer, ms
B	0.413	0.412
TRIAD-B	.352	
R	.302	.357
TRIAD-R	.247	
V	[a].079	[b].107
TRIAD-V	[a].079	

[a]Does not include 0.245-ms initial overhead.
[b]Does not include 0.328-ms initial overhead.

The vote times were found to vary with the number of processors in the configuration and the location of the task replicates in the schedule table. These variations were typically on the order of 10 to 25 percent.

The overhead due to the second category, the executive task overhead, is given below. The Interactive Consistency Tasks were the major contributors in this category and accounted for 56 percent of the executive task overhead in the optimized Version V. The overhead when a single frame is scheduled is

Version	Overhead, ms	Frame size, ms	Overhead, percent of frame size
B	60.8	83.2	73.2
R	32.0	51.2	62.5
V	28.8	44.8	64.3

The reduced major frame size for Versions R and V arises because the improved vote performance enables some tasks to be scheduled in fewer subframes.

The overhead for a triple frame is

Version	Overhead, ms	Frame size, ms	Overhead, percent of frame size
B	92.8	160.0	58.0
R	64.0	121.6	52.6
V	60.8	108.8	55.9

Concluding Remarks

The Software Implemented Fault Tolerance (SIFT) computer system requires significant overhead to achieve its fault tolerance. Several versions of SIFT—Versions B, R, and V—evolved as improvements were made to reduce this overhead. Version B

is the original delivered version of SIFT. This version only runs by disabling the clock interrupt function while many of the executive tasks are executing. This disabling of interrupts is unacceptable, since it seriously disrupts the cyclic output of the application tasks. To eliminate this problem, the system was redesigned to produce Version R. A drastic reduction in the Reconfiguration Task overhead was obtained, and feasible schedules were constructed without disabling of the interrupts. Finally, a redesign of the vote subsystem resulted in Version V.

The voting and interactive consistency functions were found to be the primary sources of operating system overhead. Unfortunately, these functions seem to be inherently expensive when implemented in software. Several modifications were made to the vote subsystem, but only moderate improvements were obtained. Even in the improved system, with as few as six input variables, the five-way vote time can consume over 30 percent of a 3.2-ms subframe. The Interactive Consistency Tasks require 13.1 ms for every iteration of the applications task set (Version V). The Interactive Consistency Tasks along with the other Global Executive Tasks consume at least 55.9 percent of each major frame. By contrast, a simplex system with no voting or redundancy management would use less than 10 percent of each subframe during scheduling. If a single communications task similar to Interactive Consistency Task 1 (IC1) is utilized, the executive task overhead is reduced to about 20 percent of a single 16.0-ms major frame or 3.2 percent of a 100-ms major frame. The fault-tolerant requirements of the SIFT system produce an overhead at least 3 times that of conventional systems. There appears to be little hope for improvement of these figures without additional hardware support.

The vote time dependency on the schedule table and on the number of working processors in the system is a serious obstacle to validation. One must be careful that sufficient processing time is allocated to a task to cover all possible configurations of SIFT in which the task may run. The validation problem is further compounded because erroneous data also increase the vote time. The marginal increase in performance gained by adding software shortcuts is offset by the increased effort required for validation. By designing the vote system so the vote time is constant and independent of the schedule table or number of working processors, the complexity is reduced and the multiplicity of test modes is eliminated. Of course, the vote time is then always worst case.

NASA Langley Research Center
Hampton, VA 23665
December 10, 1984

References

1. Wensley, J. H.; Levitt, K. N.; Green, M. W.; Goldberg, J.; and Neumann, P. G.: *Design of a Fault Tolerant Airborne Digital Computer. Volume I—Architecture.* NASA CR-132252, 1973.
2. Ratner, R. S.; Shapiro, E. B.; Zeidler, H. M.; Wahlstrom, S. E.; Clark, C. B.; and Goldberg, J.: *Design of a Fault-Tolerant Airborne Digital Computer. Volume II—Computational Requirements and Technology.* NASA CR-132253, 1973.
3. Wensley, J. H.; Goldberg, J.; Green, M. W.; Kautz, W. H.; Levitt, K. N; Mills, M. E.; Shostak, R. E.; Whiting-O'Keefe, P. M.; and Zeidler, H. M.: *Design Study of Software-Implemented Fault-Tolerance (SIFT) Computer.* NASA CR-3011, 1982.
4. Goldberg, Jack; Kautz, William H.; Melliar-Smith, P. Michael; Green, Milton W.; Levitt, Karl N.; Schwartz, Richard L.; and Weinstock, Charles B.: *Development and Analysis of the Software Implemented Fault-Tolerance (SIFT) Computer.* NASA CR-172146, 1984.
5. Lamport, Leslie; Shostak, Robert; and Pease, Marshall: The Byzantine Generals Problem. *ACM Trans. Program. Languages & Syst.*, vol. 4, no. 3, July 1982, pp. 382–401.
6. Pease, M.; Shostak, R.; and Lamport, L.: Reaching Agreement in the Presence of Faults. *J. ACM*, vol. 27, no. 2, Apr. 1980, pp. 228–234.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-86322		

4. Title and Subtitle	5. Report Date
Measurement of SIFT Operating System Overhead	April 1985
	6. Performing Organization Code: 505-34-13-32
7. Author(s): Daniel L. Palumbo and Ricky W. Butler	8. Performing Organization Report No.: L-15855
9. Performing Organization Name and Address: NASA Langley Research Center, Hampton, VA 23665	10. Work Unit No.
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address: National Aeronautics and Space Administration, Washington, DC 20546	13. Type of Report and Period Covered: Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This paper presents the results of experimentation performed in the Langley Avionics Integration Research Laboratory (AIRLAB) to measure the overhead of the Software Implemented Fault Tolerance (SIFT) operating system. During the course of this experimentation, several versions of the operating system evolved. Each version represents different strategies employed to improve the measured performance. Three of these versions are analyzed here. The internal data structures of the operating systems are discussed in sufficient detail to allow the reader to understand the experimental results and appreciate the modifications. The overhead of the SIFT operating system was found to be of two types—vote overhead and executive task overhead. Both types of overhead were found to be significant in all versions of the system. Improvements incorporated at NASA substantially reduced this overhead; even with these improvements, the operating system consumed well over 50 percent of the available processing time.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
Fault tolerance Performance Voting Scheduling Reconfiguration Interactive consistency	Unclassified—Unlimited Subject Category 62

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	26	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1985

National Aeronautics and Space Administration

Washington, D.C.
20546

Official Business
Penalty for Private Use, $300

THIRD-CLASS BULK RATE

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

NASA

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return