A Performance Evaluation of the Software-Implemented Fault-Tolerance Computer

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

The results of a performance evaluation of the Software-Implemented Fault-Tolerance (SIFT) computer system conducted in the NASA Avionics Integration Research Laboratory are presented. The essential system functions are described and compared to both earlier design proposals and subsequent design improvements. Using SIFT's specimen task load, the executive tasks, such as reconfiguration, clock synchronization, and interactive consistency, are found to consume significant computing resources. Together with other system overhead (e.g., voting and scheduling), the operating system overhead is in excess of 60%. The authors propose specific design changes that reduce this overhead burden significantly.

Introduction

THE Software-Implemented Fault-Tolerance (SIFT) computer system was developed by SRI International for NASA 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 control flight-critical functions[1,2] in the next generation of aircraft. Detailed design studies were made of fault-tolerant architectures that could meet reliability and processing requirements.[3,4] Following these studies, SRI International and the Bendix Corporation designed and built the SIFT system, which was delivered to NASA's Avionics Integration Research Laboratory in April 1982.

The basic attributes of fault-tolerant computers are:

1) Redundant hardware hardware and tasks.

2) Errors caused by hardware faults are masked by voting the redundant outputs.

3) To increase reliability, faulty hardware is removed from the system, i.e., reconfiguration.

Important distinctions between SIFT design concepts and other fault-tolerant computers are:

1) The mechanisms used to achieve fault tolerance are hidden from the application programmer, i.e., the fault tolerance is transparent.

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

3) Different tasks can be replicated to different levels (i.e., a noncritical task may be simplex, whereas more critical tasks can be replicated three- or fivefold).

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

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

6) The redundant computers are loosely synchronized.

A major objective of the SIFT design was to reduce the hardware failure rate by implementing as much of the system as possible in software (i.e., keeping the hardware component count to a minimum). This software-intensive design philosophy deliberately sacrificed performance to maximize reliability. It was implicitly assumed that failure due to software error would be eliminated by formal proof of correctness.

Received Jan. 23, 1985; revision received June 3, 1985. This paper is declared a work of the U.S. Government and therefore is in the public domain.

*Aero-Space Technologist, Fault Tolerant Systems Branch, Flight Control Systems Division.

†Aero-Space Technologist, System Valadation Methods Branch.

The SIFT design, as first described by Wensley,[1,5] embodied many desirable features, e.g., transparent fault tolerance and preemptive scheduling. However, compromises in the design, which stemmed from bus performance limitations[4] and an attempt to prove the operating system software correct,[6] resulted in a final product which differs considerably from the original intent.[7] Table 1 lists some of the discrepancies between the original SIFT concept and SIFT as it was delivered to NASA. This version of SIFT is referred to as the SIFT baseline.

At this point, the reader is cautioned not to judge the SIFT design prematurely. The SIFT baseline represents a design that was actually in the middle of its design process. Since the delivery of SIFT, development and testing have continued at NASA Langley Research Center and several versions of the operating system have evolved.[8] Each new version represents different strategies employed to improved the performance of particular areas of the operating system.

The SIFT System

The SIFT architecture consists of a fully distributed configuration of Bendix BDX930 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 are in the current system; reliability estimations have demonstrated that this is adequate to meet the stated reliability goals of failure probability of less than 10^{-9} for a 10-h flight.

The assignment of tasks to processors in SIFT is predetermined by a task schedule table defined by the application designer. As processors fail, the available hardware complement changes. Consequently, a task schedule must be defined for each configuration level the system may encounter. To accomplish reconfiguration, the SIFT operating system selects the appropriate task schedule. The decision to reconfigure is based on error information obtained when the replicated data are voted.

The synchronization of the BDX930 computers is fundamental to the correct functioning of SIFT's broadcast communication system. Interprocessor communication is completely asynchronous. No hand-shake signals or rendezvous mechanisms are used. The validity of data is dependent on the precedence established in the task schedule and the processor synchronization.

A redundant system may be susceptible to certain "malicious" process failure modes if, for example, it is necessary to distribute simplex data throughout the system. This situation would arise when each channel in the system is

connected to separate replicated sensors. In order to tolerate the failure of one of the sensors, the processors must exchange the sensor data and select an input value. Under these circumstances, it is possible for a faulty processor to make a good processor look bad, thus defeating the redundancy management system. The use of an interactive consistency algorithm[9] eliminates this failure mode.

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 provide synchronization, redundancy management (fault isolation and reconfiguration), and interactive consistency.

To achieve a realistic operating environment during testing, an autoland function has been programmed on the SIFT system. The autoland function consists of four tasks, including pitch and lateral stability augmentation routines. A total of 18 variables is voted by the application task set. An airframe simulation supports the tasks' execution. The application tasks input 21 variables and output 9 variables to the simulation. The 21 input variables must be processed by the interactive consistency tasks.

Synchronization

The synchronization of SIFT's independent processors is fundamental to SIFT's strategy of preagreed communication times. A data-producing task broadcasts its data at time T. The voter cannot access this data until T + maximum broadcast time + maximum clock skew.

$$\text{VOTE\_TIME} > \text{BROADCAST\_TIME} + B + \delta$$

where B is the maximum broadcast delay, and δ is the maximum skew between the processors.

In order for this strategy to succeed, the working processors' clocks must never drift further apart than δ. In SIFT, the voter runs 91.2 μs after the start of a subframe and $B = 18.2$ μs (see communications section below). Thus, the processors must be synchronized to within 73 μs. SIFT utilizes a decentralized clock synchronization algorithm implemented as an executive task. The algorithm contains the following steps:

1) All participating processors exchange clock values.
2) The differences (or skews) between the local and external clocks are calculated.
3) All skews above a certain threshold are set to zero.
4) The mean of the skews is used to correct the local clock.

The performance of the algorithm has been characterized by a mathematical theorem proven by SRI International.[4] This theorem describes the worst-case clock skew in terms of some measurable system parameters.

Theorem. If

$3m < N$ and

$\delta > N/(N-3m)[2\epsilon + \rho(R + 2(N-m)S/N)]$ and

$\delta > \delta_0 + \rho R$ and

$\delta << R$ and

$\delta << \epsilon/r$

then the algorithm satisfies for all nonfaulty p and q:

$$C_p^{(i)}(T) - C_q^{(i)}(T) < \delta$$

$$C_p^{(i+1)}(T) - C_p^{(i)}(T) < \delta + \epsilon$$

where

N = number of processors
m = number of faulty processors
δ = maximum clock skew
ϵ = maximum error in reading another processor's clock
ρ = maximum drift rate between two good clocks
R = resynchronization period
S = computation time for algorithm
δ_0 = initial clock skew
$C_p^{(i)}(T)$ = real time corresponding to clock time T during iteration i of processor p

Choosing values for the parameters that describe SIFT best, we have

N = total number of processors, $= 6$
m = number of failed processors, $= 1$
ϵ = maximum clock read error, $= 26 \times 10^{-6}$
ρ = maximum drift rate, $= 22.4 \times 10^{-6}$
R = major frame period, $= 0.1$ s
S = length of clock task, 2×10^{-3} s

The parameters ϵ and S were measured experimentally. S is the execution time of the clock task and was easily obtained. The

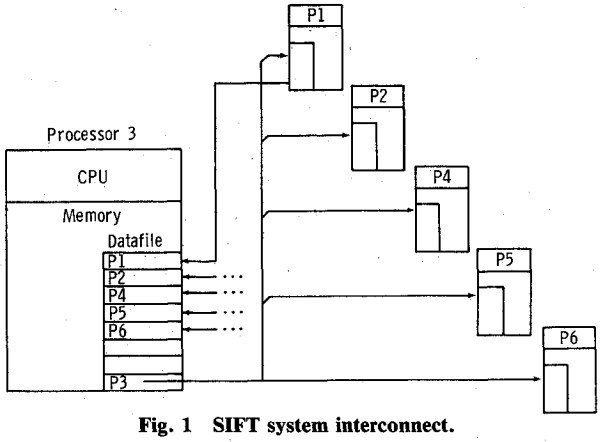

Fig. 1 SIFT system interconnect.

Table 1 Comparison of baseline SIFT to 1978 IEEE description

1978 IEEE description	Baseline SIFT
Priority-based periodic scheduling (preemptive)	Static preplanned scheduling (nonpreemptive)
Arbitrary task length	All tasks must fit in a subframe time slot
Dynamic allocation of tasks to processors	The application designer must build schedule tables and statically assign task replicates to processors.
Voting is transparent to the applications designer through use of operating system routines to obtain results	The application designer must build a vote table that corresponds to the precalculated task schedule table
Multiple-iteration rates supported	Single-iteration rates supported

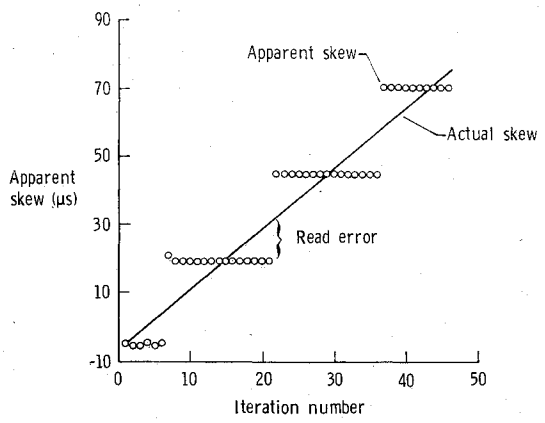

Fig. 2 Apparent clock skew.

```
Const (*global*)
.
.
QX = 103; (*datafile offset*)
.
.
```

Task producer;	Task consumer;
.	.
.	X: = Postvote(QX);
.	.
Stobroadcast(QX, 25)	.
.	.
End;	End;

Fig. 3 Interprocess communication.

measurement of read error was not as straightforward. Figure 2 shows a plot of a processor's perception of his skew relative to another processor in the configuration. This apparent skew is the sum of the actual skew and the read error. Such a graph can be obtained for every processor pair. The skew data were taken with the clock correction part of the algorithm disabled, leaving the two processors free to drift apart. Since, in SIFT, a processor cannot read another processor's clock directly, the clock synchronization task provides each processor with a window of time (114 μs) during which its local clock is repeatedly broadcast. Concurrently, all of the other processors are looping, waiting for a new clock value to arrive. As the processors drift apart, the apparent skew remains constant until data from another cycle of the broadcast loop is received. This results in the staircase shape of Fig. 2. The height of the staircase relates to the broadcast loop time of the algorithm, 26 μs.

The difficult issue is how to decouple the read error from the actual skew. One method is to measure the clock drifts externally using a global time base. A second method is to make certain assumptions about the error term and use linear regression to estimate the actual drift between two clocks. The slope of the line is the drift rate. The vertical distance of the data point from the regression line is the read error. Preliminary analysis from small data samples indicates that 26 μs is a reasonable upper bound on the read error. Further experimentation and statistical analysis are necessary before strong statements can be made about this parameter.[10]

Solving for the maximum clock skew using these parameter values we have:

$$\delta = N/(n-3m)\{2\epsilon + \rho[R + 2((N-m)/N)S]\}$$

$$= 6/3\{5.2\times10^{-5} + 22.4\times10^{-6}[0.1 + 2(5/6)2\times10^{-3}]\}$$

$$= 2[5.2\times10^{-5} + 0.23\times10^{-5}]$$

$$= 107\mu s$$

Although this figure is well above the required 73 μs, the condition has been remedied by moving the task scheduler before the voter. With this modification, the voter runs 127 μs after the start of the subframe. If further experimentation demonstrates that a larger bound is needed for the read error, the vote will have to be delayed further by adding some "deadtime" code before it. However, it is obvious that the read error is the primary component of δ because of the software implementation of the clock read function. A trivial modification to the communication interface could provide practically instantaneous recognition of a clock arrival. With this capability the clock read error could be reduced to a few instruction cycles.

Table 2 Interactive consistency overhead

Task period, ms	Execution time, ms	Overhead, %
100.0	12.9	12.9
50.0	12.9	25.8
33.0	12.9	39.1
25.0	12.9	51.6

Communications

The Broadcast Network

The internal communication architecture in SIFT is a fully connected broadcast network. Each processor in the system contains a 1K block of memory called the "data file" which serves as an I/O interface to the network. Each processor's "data file" is divided into eight "mailboxes" of 128 words each, see Fig. 1. The local processor has access to the whole data file, but an external processor can only write into the "mailbox" section allocated to it (i.e., determined by where the processors are plugged into the SHIFT chassis). The local processor uses the remaining mailbox as his output buffer. The broadcast link operates at 4 Mbps, but due to contention for the receiver's data file, a broadcast can take anywhere from 8.6 to 18.2 μs to complete. To avoid data overrun in the receiver, the hardware limits the transfer rate to about 65K words per second.

The SIFT operating system provides a primitive mechanism for task-to-task communication. Data generated by a task are made available to other tasks only after termination of the data-producing task. There is no facility for interprocess communication between concurrent processes in the system. A task outputs a data value by calling the system service STOBROADCAST(bufnum, value). Bufnum is a constant that holds the location in the processor's mailbox where the value is to be stored. Input data are made available in the array POSTVOTE[bufnum] (see Fig. 3). The executive references several data structures to complete the transaction from mailbox to POSTVOTE array. First, a list of output buffer numbers the task will produce must be placed into the buffer information table (BINF array). Second, the task must be assigned a slot in the schedule table. Finally, the subframe in which the data is to be voted must be indicated in the vote schedule tables. In the SIFT baseline, these tables are constructed manually by the application designer. Obviously communication can be accommplished in this manner, however, several disadvantages are evident:

1) There are no safety features to ensure that the application designer does not assign two "buffer" constants the same data-file location.

2) There are no consistency checks to ensure that data buffers actually scheduled for voting have been STOBROADCASTed.

3) Task replication is directly visible to the application designer—he must coordinate the vote with the task replicates. Recently, an automated schedule generator has been developed that simplifies this procedure mitigating most of these objections (see section on scheduling).

Interactive Consistency

Data transfer between SIFT and external devices is done via a MIL-STD-1553A bus. Each processor is equipped with a 1553A interface, however, under task schedule control, only three processors participate in external communications at any time. Since all processors must receive identical input data, SIFT utilizes a set of interactive consistency tasks to distribute the data. These tasks receive inputs from replicated sources such as sensors over the 1553a bus and distribute these inputs to all processors. They ensure that all processors get identical values even in the presence of any arbitrarily malicious behavior by a single processor. The tasks run at the sampling rate of the input data stream (presently about 30 Hz). Their execution time is 12.9 ms. Since the tasks must be run at the same frequency as the input data, the total overhead varies with this rate (see Table 2). If input data are received every 33 ms, system overhead attributed to interactive consistency alone is 39%.

Scheduling

Task scheduling in SIFT is nonpreemptive and based on precalculated schedule tables. In the baseline SIFT design, a schedule table consists of 32 3.2-ms subframes which comprise a 100-ms major frame. Tasks are statically allocated to these subframes. The application designer must build a schedule for each processor at every level of configuration, allocating all of the task's replications to the appropriate system processors.

Clearly SIFT has changed significantly from the original description (see Table 1). Most of the changes can be attributed to simplifications made to enable a formal proof of correctness. There are several advantages to the nonpreemptive approach chosen for SIFT. First, the resulting system is simple and deterministic. Not only does this facilitate a formal proof of correctness, but also simplifies other approaches to validation, such as testing. Another important advantage is that static structures can reside in read-only memory providing greater immunity to transient faults. The static schedule, vote, and task tables of SIFT could be placed in ROM minimizing the region of vulnerability. In SIFT, only 5K words of dynamic memory are unrecoverable and hence vulnerable to transients.

Several disadvantages of the nonpreemptive approach are evident.

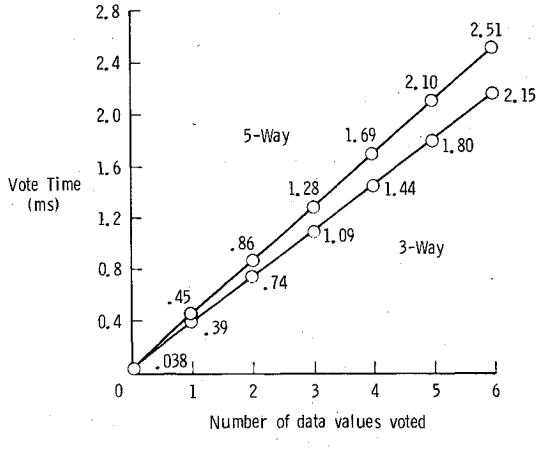

Fig. 4 Baseline SIFT vote overhead.

1) Response to asynchronous events must be accomplished by continually "polling" the event by an active process. Such an approach has a built in latency time and is clearly inefficient.

2) Variable task-execution times result in inefficient utilization of the CPU since the schedules must be built for "worst-case" performance.

3) There is no flexibility in the system to provide for changing external conditions such as flight phases or temporary system overloads. Hence, the applications must be built with an inordinate number of Boolean variables (often called mode logic or discretes) so that the system can be responsive to phase changes. Such Boolean logic increases the variability in task execution time which compounds the CPU utilization problem mentioned previously.

4) A preemptive schedule can always be constructed which is shorter than a nonpreemptive schedule for a multiprocessor system.

5) The schedule tables must be rebuilt whenever a new task is added, even if it is a low-priority task.[5]

While the current version of the SIFT system retains a nonpreemptive scheduler with the advantages and disadvantages as stated above, two features have been added that simplify the scheduling process and make the task schedule more efficient. First, the size of the subframe can be variable length (any number of 1.6-ms clock ticks). A minor hardware modification can reduce the granularity of a clock tick to tens or hundreds of microseconds. This modification would greatly improve the efficiency a static task schedule can attain. Second, a schedule generator is now available which reduces the effort necessary to specify the SIFT task and vote schedules, and buffer information tables.[11]

The schedule generator does not hide the fault-tolerance mechanisms entirely. The application designer is still responsible for the scheduling of the global executive tasks as well as the application tasks. Although this procedure is not ideal, the risks involved are reduced by providing a template containing recommended global executive scheduling. The replication levels of the application tasks, i.e., five-way, three-way, or simplex, are the designers' only other input to the redundancy management procedure.

Voting and Error Detection

The voter is driven by the vote schedule, which designates the buffer numbers of the data to be voted in a subframe. For each buffer number, the voter uses pointers in the Buffer Table or "BT array" (assembled by the reconfiguration task described below) to search the data file for valid data. Once the redundant values have been fetched, they can be voted. The voter can perform five or three-way voting which may require a search of all eight sections of the data file. Since the search consumes most of the vote time, a three-way vote takes only slightly less time than a five-way vote. The overhead is a linear function of the number of data values to be voted (see Fig. 4).

A five-way vote takes about 413 μs in the absence of errors. Hence, with four data values, the vote overhead would take over 50% of the 3.2-ms subframe. If an error occurs, an error count associated with the processor that produced the bad data is incremented. This function adds 15 μs to each vote. The application designer must consider this potential extra delay to ensure his tasks do not exceed the subframe boundary.

In the current version of SIFT, the voter has been modified to vote a task's entire set of output data per invocation, Fig. 5. This was done to reduce the overhead that results from data lookup. Although a larger penalty is paid to vote one buffer—450 vs 413 μs—each subsequent vote is more economical. One disadvantage of this method is that the vote of a task's data cannot be spread over many subframes, e.g., if a task has 10 output buffers, it would put an inordinate burden on the subframe in which its vote occurs.

Reconfiguration

The reconfiguration task is scheduled at the end of a major frame. Although a 3.2-ms subframe is allocated to the reconfiguration task, in the baseline SIFT system, the task is idle for most of the time. When a configuration change is dictated by the fault-isolation task, the reconfiguration task assumes two major responsibilities. First, it searches the task tables for the set of schedules that corresponds to the new configuration. Second, it builds a new BT array based on the new schedules. The BT array indicates which mailbox (and therefore which processor) will contain valid data when the vote takes place. The exact time for reconfiguration depends on the number of working processors; however, the worst case is 35.19 ms or 11 subframes (see Fig. 6).

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. This condition could not be tolerated and several new algorithms were tested in AIRLAB. The most successful approach was to modify sections of the local and global executives to use virtual processor numbers. The use of virtual processor numbers allows the BT array to be precalculated for every configuration level. It is left to the reconfiguration task to select the correct schedule as before, plus construct a mapping between the virtual and real processor numbers. This technique reduced the execution time to 2.5 ms, which fits comfortably within the 3.2-ms subframe.

Total System Overhead

In order to put the total system overhead in perspective, the following operational mode is assumed. A major frame contains three iterations of the application tasks and, therefore, requires three iterations of the interactive consistency tasks. To accomplish this, the major frame is divided into three sections 35.2-ms long, or 22 slots (a slot is one 1.6-ms clock tick). Therefore, the major frame is 105.6 ms. Within each section, 18 slots are dedicated to the interactive consistency and application tasks. The remaining four slots provide sufficient processing time for clock synchronization and redundancy management tasks to complete one iteration every major frame. Table 3 lists processor utilization summarized under the categories of dispatch overhead (voting and scheduling), and global executive and application tasks. In this configuration, the SIFT system achieves a utilization of 79%. However, 80% of this processing is dedicated to dispatch overhead and the global executive.

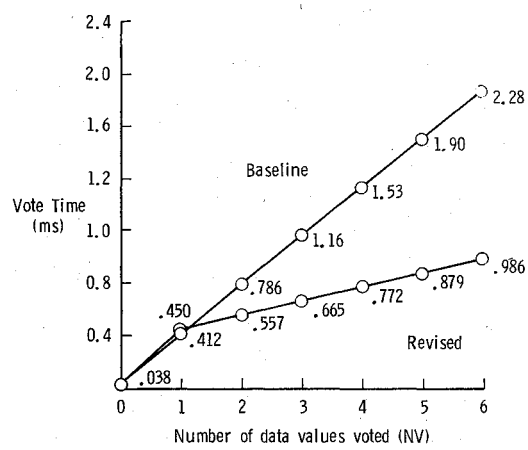

Fig. 5 Baseline vs revised system vote overhead.

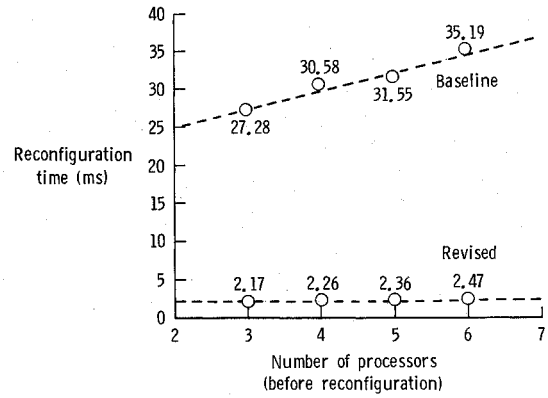

Fig. 6 Reconfiguration times.

Table 3 Summary of SIFT utilization

System function	Processor time, %
Dispatch	21.6
Global executive	41.6
Application tasks	15.8
Idle	21.0

Evaluation of the SIFT Concept

There are many worthy aspects of the SIFT concept. It is unfortunate that, to distinguish SIFT from other fault-tolerant designs (e.g., the Fault-Tolerant Multiprocessor—FTMP[12]), the emphasis is often placed on software vs hardware voting, when, in the authors' opinion, the primary distinction lies in the fact that SIFT reconfigures task schedules and not hardware devices. This approach has been shown to be simple and efficient. Simplicity is significant because, with all of the software implementation, the entire SIFT operating system is just 1500 lines of Pascal code. The small size of the operating system and straightforward algorithms lend themselves well to validation efforts—the most difficult aspect of fault-tolerant design. A second, but no less important, difference is the loose synchronization of the SIFT processors. The SIFT synchronization algorithm functions over standard communication paths allowing fully distributed, nonhomogeneous processors to synchronize.

These two distinguishing characteristics—the reconfiguration of task schedules and loose synchronization—have a major impact on system overhead. Both characteristics complicate the vote procedure which is performed during task-to-task communication and interactive consistency. Due to loose synchronization, a certain amount of delay must follow communication before voting can take place. If normal processing does not provide a sufficient delay, "deadtime" must be added. Because the task schedule is different for different configurations, a "lookup" procedure must be used to locate and retrieve the data before each vote. This technique requires more overhead than a tightly coupled system, such as FTMP, where data can be voted and error information gathered during the communication process.

If the SIFT concept were used in a production system today, the software-intensive design philosophy should be reconsidered. Recent advances in the fabrication of electronic devices have altered the nature of the performance/reliability tradeoff. An increase in circuit complexity can be accommodated without a significant increase in failure rate. Furthermore, the use of hardware implementation does not preclude proof of correctness. In fact, algorithms implemented in hardware are often easier to verify than their software counterparts.[13]

Three functions of the SIFT operating system can benefit directly from hardware support. They are the implementation of the synchronization algorithm, the vote algorithm, and interactive consistency. As stated previously, a large clock read error is associated with the software looping inherent in the

current implementation of the synchronization algorithm. The read error, and, therefore, the upper bound on the system clock skew, can be reduced significantly with minor hardware support. Although loose synchronization and task schedule reconfiguration complicate the vote algorithm, it is nevertheless possible to implement this algorithm efficiently in hardware. In a hardware implementation, the designer could exploit parallelism in the data lookup and vote procedures to achieve significant performance improvement. Optimally, the vote algorithm would be added as a special function of the communication hardware. Interactive consistency would benefit directly from an increase in vote performance. The remaining contributor to interactive consistency overhead is the time required to redistribute data. Again, special functionality built into the communication hardware could reduce this burden. Because all of the suggested hardware support is integrated into the communications interface, SIFT would retain its characteristic of using "off-the-shelf" computers to construct fault-tolerant systems.

Conclusions

Software-implemented fault tolerance places a tremendous overhead on a system. In the test case considered, over 60% of the processor's time is devoted to operating-system overhead. While it can be argued that future processors with perhaps 10 times the speed of the BDX930 would reduce this overhead to acceptable levels, most of the overhead is proportional to the amount of data produced, and it is unreasonable to assume future systems will have an equal or lower data volume.

However, it has been shown that 100% software implementation is unnecessary to achieve the primary benefits of the SIFT architecture, i.e., loose synchronization and task schedule reconfiguration. Additional hardware functionality built into the communications interface would significantly reduce the overhead of the vote and interactive consistency processes.

Although software implementation of the redundancy management functions has imposed a large overhead, the software nature of SIFT allows speedy testing of new algorithms and methodologies. Thus, SIFT serves well as an experimental test bed for fault-tolerant systems research.

References

[1]Wensley, J. H. et al., "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," *Proceedings of the IEEE*, Vol. 66, Oct. 1978, pp. 1240-1255.

[2]Goldberg, J., "SIFT: A Provable Fault-Tolerant Computer for Aircraft Flight Control," *Proceedings IFIP Congress '80*, International Federation for Information Processing, Tokyo, 1980, pp. 151-156.

[3]Weinstock, C.B., "SIFT: System Design and Implementation," *The 10th International Symposium of Fault-Tolerant Computing*, Oct. 1980, pp. 75-77.

[4]Goldberg, J. et al., "Development and Analysis of the Software Implemented Fault-Tolerance (SIFT) Computer," NASA CR-172146, June 1983.

[5]Wensley, J. H. et al., "Design of a Fault-Tolerant Airborne Digital Computer. Volume I—Architecture," NASA CR-132252, 1973.

[6]Levitt, K. N. et al., "Investigation, Development, and Evaluation of Performance Proving for Fault-Toleant Computers," NASA CR-166008, Aug. 1983.

[7]Butler, R. W., An Assessment of the Real-Time Application Capabilities of the SIFT Computer System," NASA TM 84482, April 1982.

[8]Palumbo, D. L. and Butler, R. W., "Measurement of SIFT Operating System Overhead," NASA TM 86322, April 1985.

[9]Pease, M., Shostak, R., and Lamport, L., "Reaching Agreement in the Presence of Faults," *Journal of the Association for Computing Machinery*, Vol. 27, April 1980, pp. 228-234.

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

[11]Green, D. F., Palumbo, D. L., and Baltrus, D. W., "Software Implemented Fault-Tolerant (SIFT) User's Guide," NASA TM 86289, Aug. 1984.

[12]Smith, T. B. et al., "A Fault-Tolerant Multi-Processor Architecture for Aircraft," NASA CR 3010, July 1978.

[13]Rennels, D. A., "Fault-Tolerant Computing Concepts and Examples," *IEEE Transactions on Computer*, Vol. C-33, Dec. 1984, pp. 1116-1129.