Software Implemented Fault Insertion: An FTMP Example

Edward W. Czeck

Zary Z. Segall

Daniel P. Siewiorek

Electrical and Computer Engineering

Carnegie Mellon University

Schenley Park

Pittsburgh, Pennsylvania, 15213

15 January 1987

This paper is submitted for publication, and is presented here for early distribution.

This Research was sponsored by the National Aeronautics and Space Administration, Langley Research Center under contract NAG-1-190. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA, the United States Government or Carnegie Mellon University.

Table of Contents

List of Figures

Abstract

This report presents a model for fault insertion through software, describes its implementation on a fault tolerant computer, FTMP, presents a summary of fault detection, identifications, and reconfiguration data collected with software implemented fault insertion and compares the results to hardware fault insertion data.

The software fault insertion model assumes faults manifest to data errors at the output of a task. The implementation of the software fault insertion model on FTMP allows inserted faults to emulate faults in the processor data path, processor control path, system memory, and system transmit bus.

The experimental results showed detection time to be a function of time of insertion and system workload. For the fault detection time there was no correlation between software inserted faults, and hardware inserted faults; this is because hardware inserted faults must manifest to errors before detection, whereas software inserted faults immediately exercise the error detection mechanisms. Fault identification time for FTMP is a function of the number of modules to which the errors can be attributed. From the data, hardware inserted faults manifest to error patterns attributed to unique modules, whereas single software inserted faults were attributed to multiple sources, thus exposing the non-unique mapping between hardware and software inserted faults. Fault reconfiguration times were comparable for both hardware and software inserted faults.

In summary, although software fault insertion does not map directly to hardware fault insertion, experiments indicate software fault insertion as a means to characterize the fault handling capabilities of a system in error detection, identification, and error recovery.

1. Introduction

Validation procedures, such as those proposed in [NASA 79a, NASA 79b] include steps to characterize and evaluate the behavior of a system under faulty conditions. The means for these evaluations include the following:

1. *Computer Simulation:* Computer simulation evaluates the manifestation of faults and the system's response. The simulation models range from the Processor-Memory-Switch level through the Instruction Set Processor level, the Register Transfer level, the gate level, and to the device level. The drawbacks to simulation are the high cost of model development, computational needs, and the difficulty in model validation.

2. *Physical Fault Insertion:* Physical fault insertion places faults in the hardware of a realized system. The advantages over computer simulation include speed and fidelity to actual system faults. The drawbacks to this method are two fold. First faults insertion requires physical manipulation of components, a time consuming effort. Second the faults are limited to pin level insertion. As realizations moves from SSI/MSI to VLSI, the fault insertion level moves from gate level to system level.

This paper discusses *Software Implemented Fault Insertion*, in which hardware or physical faults are emulated by modifying program data or control. The motivations for software fault insertion include speed and automation advantages. In addition software inserted faults are repeatable within a system and across architectural and implementation boundaries. Finally, the gap between pin level fault insertion in VLSI and software fault insertion is narrowing and may be approaching equivalence.

The literature abounds with prior work demonstrating the benefits of fault insertion and the feasibility of software fault insertion at the architectural or bus level; a sampling of this prior work includes:

- The FTMP evaluation used pin level (gate level) stuck-at or inverted permanent faults. Observations noted in [Draper 83a] include the difficultly caused by incorrect functioning of the test module with the test equipment connected and damage to CMOS circuitry caused by incorrect handling. From the fault insertion experiments, they were able to evaluate the fault detection, isolation, and recovery times. Results also showed preliminary but inconclusive data on the fault coverage of FTMP. This experiment demonstrates the value of using fault insertion for fault tolerant system evaluation.

- [Schuette, et al. 86] inserted transient or soft faults in a MC68000 to evaluate software triple modular redundancy and a signature instruction stream monitor. The MC68000 realization did not allow gate level fault insertion, hence faults were

inserted on the address, data, and control bus lines. This experiment shows fault insertion at the bus level can be used to evaluate fault tolerant techniques.

- The Sperry UNIVAC 1100/60 [Boone et al. 80] has a built-in fault insertion capability to verify fault detection, isolation, and recovery mechanisms. This capability is activated during system idle time and can insert faults in the processor, memory, and I/O unit. The UNIVAC 1100/60 system shows fault tolerant mechanisms can be verified using software control at the system level.

- [Yang et al. 85] inserted faults into the iAPX 432 to evaluate software implemented triple modular redundancy. Faults were inserted by altering bits in the program or data areas of memory using the debugger. The experiment shows fault insertion may be accomplished by altering bits in the memory.

This paper is divided into five sections. The second section gives an overview of the FTMP architecture with emphasis on the fault-handling mechanisms. Section 3 describes a model for a fault tolerant system, a model for fault insertion at the architectural level, and the implementation of this model on FTMP. Section 4 presents data from software fault insertion experiments and provides a comparison, where applicable, to similar hardware fault insertion experiments. The last section concludes the paper with an evaluation of software fault insertion techniques.

2. FTMP Architecture

This section presents an architectural description of FTMP, the target machine for the software implemented fault insertion. Four subsections include a general overview, followed by a detailed description of fault detection, fault isolation, and fault recovery mechanisms

2.1 General Overview

The Fault-Tolerant Multiprocessor, FTMP, is a hardware redundant multiprocessor designed for ultrareliable avionics environments [Hopkins et al. 78, Draper 83b, Draper 83c]. The architecture, as seen by the programmer, consists of three virtual processors with local memory, connected via a common bus to global memory and I/O ports. Reliability is attained through hardware redundancy. Each virtual processor consists of a processor triad. The memory and buses are also triplicated. Spare processors, memories and buses shadow (i.e. execute the same code as) the active units, but do not participate in voting. Each triad executes synchronously and a hardware vote occurs during data transfers. The voting is performed by each receiving unit from data transferred over independent buses.

The bus structure consists of four sets of serial buses each quintuply redundant of which three are active at any given time. The buses are: the *Poll Bus* which is the bus arbiter; the *Transmit Bus* which carries addresses and data information from the processor; the *Receive Bus* which carries data from global memory or I/O ports to the processors, and the *Clock Bus*, which carries clock signals to each processor to maintain system synchronization.

FTMP employs a realtime operating system with a basic dispatch period of 40 milliseconds, referred to as Rate-4. There are two lower rate groups, Rate-3 with a period of 80 milliseconds, and Rate-1 with a period of 320 milliseconds. Lower rate tasks include application tasks and also system tasks such as the system configuration controller (SCC), a memory checker, status display and self tests.

2.2 Fault Detection

The fault detection mechanism for FTMP employs hardware voters residing at the receivers of each bus set. Disagreements at the voters set error latches, associated with each individual bus line. SCC, running as a Rate-1 task, reads the error latches to check for errors and

potentially faulty units. If an error is detected, the time of the error is stored and the fault identification routine is called. A time line of the events occurring in fault detection, identification, and reconfiguration is shown in Figure 2-1.

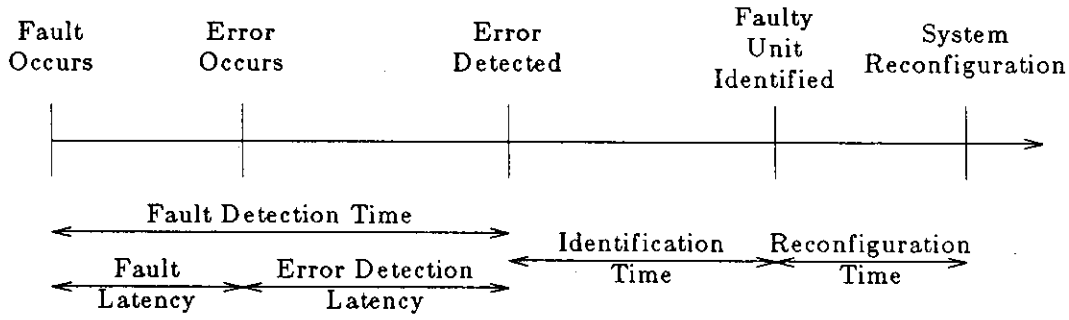

Figure 2-1: Time Line of Error Detection, Identification and Recovery

2.3 Fault Identification

The goal of fault identification is to determine from the error latch information which unit caused the error. Since an error on one bus may be attributed to multiple sources (each unit enabled on the bus), the general procedure of the fault identification routines is:

1. Determine the possible sources of the faults from the error latch information. If there is more than one source, the bus assignments are switched and the identification routine waits a Rate-1 frame for another error to occur.
2. If another error occurs, its possible sources are identified and intersected with the previous possible sources. If the new set is not unique, this step is repeated after switching bus assignment again. If an error does not occur, a transient fault analysis routine assigns demerits to all possible sources.

The fault identification routine runs as part of SCC hence the identification time will be a function of the number of passes needed to identify the fault.

2.4 System Reconfiguration

The system reconfiguration procedure entails removal of faulty units either by swapping with a spare unit or by graceful degradation. These procedures are described as follows:

1. If there is a spare unit (Processor, Memory, or Bus) and it is shadowing the faulty unit, the bus assignments are changed to bring the spare unit active and the failed unit inactive.
2. If the spare processor or memory is shadowing a triad other than the one containing the faulty unit, the spare is first brought to shadow the triad, and then the spare and failed unit are swapped.

3. Finally if there are no spare processors, the triad is retired with its good processors assigned as spares. When memories or buses fail without spares, the triad reduces to a duplex.

The Rate-4 dispatcher executes the reconfiguration commands from the information supplied by the fault identification routine. The error reconfiguration time is defined as the time from the fault identification to the execution of the reconfiguration commands.

3. Software Implemented Fault Insertion

This section describes a model for a fault tolerant system, and a model for software implemented fault insertion for a realtime operating system. The realization of the software fault insertion is presented on the example system, FTMP.

3.1 Fault Tolerant System Model

Fault tolerant systems generally use either hardware or time redundancy to achieve reliability. Under each of these schemes, there are confinement regions (hardware or time) which localize the corruption caused by a fault. Associated with the region, usually at the boundary, is an error detection and isolation mechanism (EDI) which limits fault propagation. The EDI also generates a status showing the condition of the region or system. Figure 3-1 presents a system model, based on fault confinement regions, where the regions are processors, P, memories, M, and I/O units, interconnected via buses through the EDI interface.

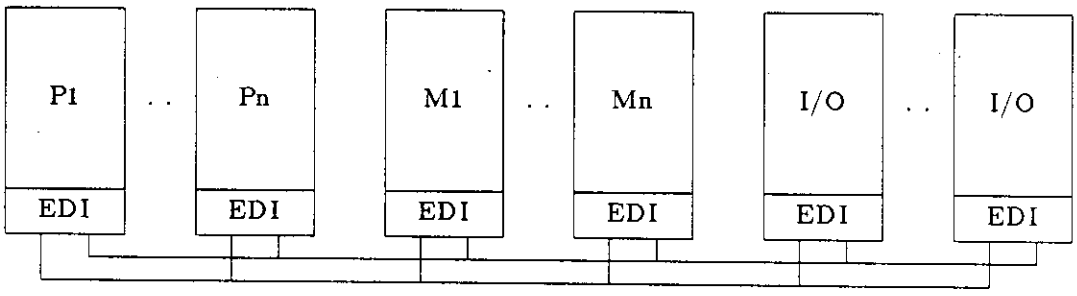

Figure 3-1: Fault Tolerant System Model

The goal of software implemented fault insertion is to force the system to appear faulty by exercising one or more of the EDI interfaces by one of the following means:

1. Immediate activation where the EDI is exercised by an error at the boundary of a confinement region, or
2. Latent activation where faults are seeded within the confinement regions.

A comparison between software fault insertion and hardware fault insertion includes:

- The goal of both fault insertion schemes is to exercise and evaluate the fault-handling mechanisms of the system.
- Software faults may be better in triggering a specific error, which is difficult to generate or reproduce with physical fault insertion.

- Physical fault insertion may be more analogous to actual faults generated in the system.

3.2 Software Fault Insertion Task Model

A model for a computational task is shown in Figure 3-2a. Data (sensors) are read at the start of the task, operations are performed on the data, and the results are written (to actuators). A fault occurring in the task would manifest to an error in the output of the results. These errors include incorrect data, no data, or late data. Hence faults in the task could be modeled as an error in the output part of the task, Figure 3-2b. Realtime execution could be modeled as a series of computational tasks with the dispatcher executing between the tasks as shown in Figure 3-2c. Adjusting the task model to fit the multiple execution rates of FTMP, let L be a lower rate task, where L is all the non Rate-4 tasks[1] concatenated together to form a single task. The L task executes at the end of the Rate-4 tasks and is interrupted by the start of the next Rate-4 frame, Figure 3-3; thus, the amount of execution time per Rate-4 frame for the L task depends on the Rate-4 frame size and the execution times of the task and dispatcher.

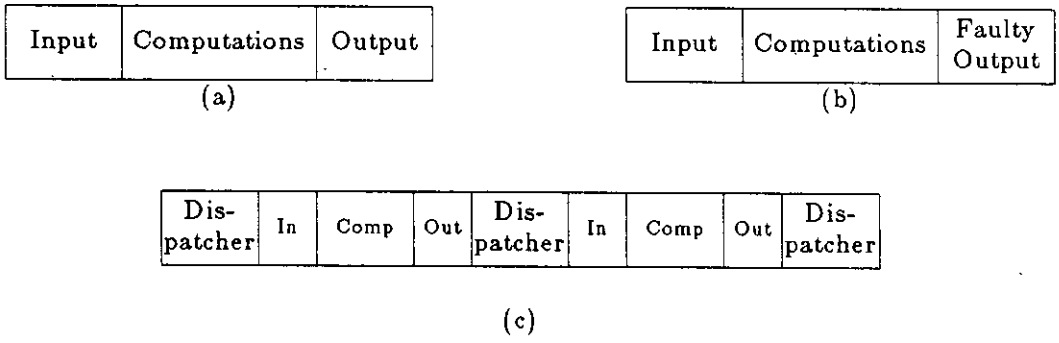

Figure 3-2: Computational Task Model

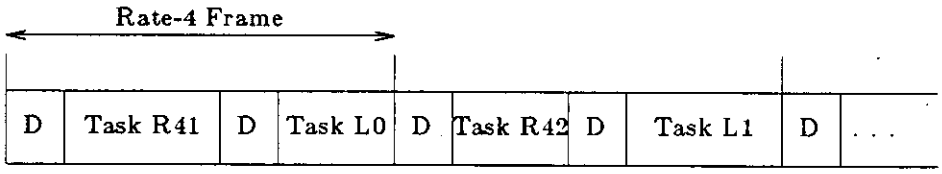

Figure 3-3: Lower Rate Task Execution Model

[1]The lower rate tasks include a clock update, System Configuration Controller (SCC), memory checker, and status display which execute at 3.125 Hz, one-eighth of the Rate-4 tasks.

3.3 Software Fault Insertion Realization

The abilities of software implemented fault insertion or of any fault insertion in general are the following:

- Location of Fault: Insertion of faults should be able to model true faults which can occur throughout the system hardware.
- Timing of Faults: A fault may occur throughout any execution task of the system; a fault insertion environment should allow similar conditions.
- Duration of Faults: Real faults are classified as either transient, intermittent, or permanent [Siewiorek and Swarz 82]; the fault insertion should allow the duration of inserted faults to vary accordingly.

The realization of the software fault insertion is unfortunately limited by the controllability of hardware in FTMP.

3.3.1 Location and Generation of Faults

The fault insertion environment must be able to insert or emulate faults in different locations. The software fault insertion environment allows faults in four regions; these regions and the means which the faults are inserted are described as follows:

- *Processor Data Path Faults:* Faults occurring in the data path may manifest to a number of different error types. These include transmission of incorrect data, no data or late data. The software fault insertion environment assumes processor data path faults manifest to incorrect data being transmitted by the processor, causing an error on the transmit bus assigned to the processor. The incorrect data is a single word, and the processor state remains good after the transmission of the bad data.

- *Processor Control Path Faults:* Faults within the control path may manifest to many different error types. These include no data transmitted, early or late data transmitted, or incorrect data transmitted. The software fault insertion environment emulates faults within this region by having the processor execute an infinite loop, resulting in no transmission of data. This causes errors on two of the buses to which the processor is assigned: the poll bus because the processor never requests the bus, and the transmit bus because no data is transmitted.

- *System Bus Faults:* Faults on a bus may be attributed to many sources, such as noise or a unit transmitting out of protocol. Software fault insertion emulates bus faults by having a processor transmit bad data on a specific bus, although the processors are generating the errors, the errors map to a particular bus.

- *Global Memory Faults:* Memory faults may be attributed to decaying bits, stuck-at bits, or incorrect address decoding. Memory faults are emulated by writing bad data into one memory module of a triad, and then performing a read of the location.

A few comments on the fault insertion are in order. First all inserted faults cause an immediate error; there is no latency between insertion and error generation. Second, the faults are transient and cause no change in processor state or corruption of data, except for the control path fault. Third, the present software implemented fault insertion environment does not exclude the later addition of latent faults. For example, local or global memory can be corrupted without an immediate read, then the detection time, the time from the change in data (fault) to the error can be measured.

3.3.2 Timing of Faults

Faults may occur at any time within the execution of a program. The model assumes faults manifest into errors in the output portion of the task. The implementation of software fault insertion allows faults to be generated in the output of Rate-4 application tasks. The occurrence of a fault is specified to a particular Rate-4 frame, but not to the time within the frame. Furthermore, faults may only occur in Rate-4 application tasks and not within the dispatcher or lower rate tasks. This limits the insertion time of the faults to specific tasks. However, the error detection mechanism for FTMP cannot distinguish the insertion time to specific tasks.

3.3.3 Duration of Faults

Faults can be transient soft, due to a temporary random environmental condition or permanent hard, due to a physical change in the hardware. The software fault insertion environment generates transient faults, and to emulate permanent faults, a transient fault is repeatedly inserted in consecutive Rate-4 frames. This gives the appearance of a permanent faults when view from the error detection and identification mechanisms.

3.3.4 Workload

A system's workload is its set of inputs received from its environment; a desirable feature within any computer evaluation environment is a controllable workload. [Feather et al. 85] developed a synthetic workload[2] generator for FTMP which was modified to include software fault insertion. The synthetic workload provides a means of specificing the following factors:

- System Configuration which defines the number of processor triads and spares.
- Number of Tasks for each rate group, and the inclusion of the system tasks, such as SCC and Status Display.

[2] A synthetic workload exercises a computer system by modeling its natural workload with generic inputs and tasks.

- Workload for each task, which includes the amount of I/O and computations per task.

3.3.5 Recovery Mechanism

In order to repeat the fault insertion experiment in gathering a large data base, a recovery mechanism must augment the software fault insertion environment. Draper Labs modified the system configuration controller to repair and activate processor 3 before fault insertion.[3] This repair code was modified allowing for the repair and activation of the last unit failed (processor, memory, or bus), before each fault insertion.

3.4 Experimental Environment

The experimental environment for software fault insertion can be divided into three sections: the experimental set-up, the collection of data, and the analysis of data. This section describes these areas on the FTMP implementation.

3.4.1 Parameters

The first phase of the experimental procedure is experimental setup and selection of parameters. A program queries the user on the selection of the parameters, and from the inputs, generates a command file which properly configures FTMP and collects data. The controllable parameter include:

- Workload: The system workload includes the amount and distribution of tasks between the rate groups, the amount of I/O and computation executed by each task, the inclusion of system tasks, and the overall system configuration.
- Location: The different locations for fault insertions are described in Section 3.3.1.
- Timing: The time of fault insertion is controlled by the Rate-4 frame, hence a 40 millisecond resolution.
- Duration: Either a transient (single) fault or a permanent (repeated) fault may be inserted, as described in Section 3.3.3.
- Data Collected: The data which may be collected includes: the application tasks' execution time; the fault insertion, detection, identification, and reconfiguration time; the identification of the failed unit; and the reason code for the failure.

[3] Draper's hardware fault insertion system allowed faults to be inserted in processor 3, hence Draper's software checked status and activated processor 3 before inserting faults.

3.4.2 Experimental Execution

The second phase of the experimental procedure is insertion of faults and the collection of the data. During each experimental loop the following actions occur:

1. The system repairs any module which failed during the last cycle.
2. The fault inserter is started and the workload data collection cycle begins. Workload data (task execution time) is collected for one Rate-1 frame, and the inserted faults trigger the fault-handling mechanisms whose execution times are also collected.
3. The requested data is uploaded to the host and the cycle repeated.

3.4.3 Data Analysis

The third phase of the experimental procedure is data analysis. The data analysis program takes the absolute timer values collected from FTMP and records differences between two events as requested by the user. The average, standard deviation, minimum, maximum, and a histogram of the data is then printed.

4. Experiments

With Software Fault Insertion implemented, the next step was to run experiments evaluating the abilities of the environment. The experiments exercised most of the parameters available in the software fault insertion environment. In particular the location of the fault, time of insertion, and system configuration were the primary parameters varied. The data collected from these experiments involve a measurement of the system workload, and the times of fault insertion, fault detection, fault identification, and fault recovery. Additionally, the unit which failed and the reason code for the failure were stored for analysis of missed diagnosed faults. This section details the experiments performed. Comparisons to hardware fault insertion results [Draper 83a] are made where appropriate.

4.1 Summary of Draper's Fault Insertion Data

Draper under contract to NASA completed extensive hardware fault insertion experiments at the pin (gate) level [Draper 83a]. This section summarizes their experiments and presents a comparison between Draper's hardware fault insertion and the software fault insertion experiments.

Draper's experiments inserted faults at the pin level of the processor; the faults were single bit stuck-at zero, one, or inverted. The data is divided by the fault location, where the locations are cards in the LRU's.[4] For each card, several chips were pulled and faults inserted on each of the chips. For our comparison data from four different card was taken: the CPU data path card (CPUD); the CPU control path card (CPUC); the bus interface transmit card (BIT); and the cache controller card (CC). These correspond to the software fault insertion locations of data path, control path, transmit bus, and data path respectively; Figure 4-1 diagrams a possible mapping between hardware and software inserted faults.

The parameters for Draper's data was many times unspecific and for the purpose of comparison to the software inserted faults some assumption were made:

- The time of fault insertion was random for Draper's data, whereas with the software fault insertion, the insertion time was specified to the output portion of a Rate-4 task.

[4] An LRU is a Line Replaceable Unit, each identical and containing a processor, memory and the necessary bus interface circuitry.

- The system workload was unknown, but we will assume a light workload with the Rate-4 frame size at 40 milliseconds for Draper's data. The workload for the software insertion was one Rate-4 task, one Rate-3 (timer) task, and three Rate-1 tasks (display, SCC, and readall), and the Rate-4 frame size was stretched to 50 milliseconds; hence the Rate-1 frame size was 400 milliseconds. The difference in frame sizes for the two insertion methods should increase the time measurements for the software inserted faults, approximately 25%, in comparison to the hardware fault insertion measurements.

- The system configuration for Draper's data was unknown, a reasonable assumption is three triads executing with either zero or one spare processor. The software data lists the configuration either as two or three triads without spare, or two triads with spare.

Draper's Hardware Fault Injection Location	Possible Fault Manifestations	Software Fault Insertion Comparison Location
CPU Data Path Card	Bad Data from Processor	Data Path
	No Data from Processor	Control Path
CPU Control Path Card	Processor Hangs	Control Path
	No Data from Processor	Control Path
	Bad Data from Processor	Data Path
Cache Controller Card	Processor Hangs	Control Path
	No Data from Processor	Control Path
	Bad Data from Processor	Data Path
Bus Interface: T-Bus Card	Bad Data on Bus	T-Bus
	No Data on Bus	T-Bus

Figure 4-1: Possible Fault Mapping Between Hardware and Software Inserted Faults

4.2 Fault Detection Time

Fault detection time is the time from the insertion of a fault until an error is detected by the system. For software inserted fault, the insertion time is at the end of a specified frame, whereas with Draper's hardware inserted faults, the insertion times are any point within the frame. Error detection, reading of the error latches, is done by SCC at Rate-1. Hence the detection time for software inserted faults should be a maximum of one Rate-1 frame (400 milliseconds), the latency in reading the error latches. For hardware inserted faults the detection time will include the manifestation of the fault into an error, along with the delay in reading the error latch.

In predicting the detection time for software inserted faults, the parameters affecting the detection time are:

- Workload: A large workload stretches the frame size, placing the detection point later in the frame. Likewise, a large workload limits the execution time per Rate-4 frame of the lower rate tasks (e.g. error detection). The workload function is expressed by $R4task$, the Rate-4 **task size**, and $R4Frsize$, the Rate-4 **frame size**, both measured in milliseconds.
- Time of error detection: The point which error detection occurs within the realtime cycle affects the latency from the time of fault insertion. This is determined by the amount of time which the lower rate task executes before the error detection routine is run. This time is represented as $LDet$ and measured in milliseconds.
- Time of Insertion: The of point of fault insertion within the realtime cycle in conjunction with the time of error detection governs the fault detection latency. The time of insertion is represented as Tin and measured in Rate-4 frames.

Finally let: $LxTime$ be defined as the amount of time which the lower rate tasks execute per frame, where $LxTime$=max ($R4FrmSize$–$R4Task$,10) milliseconds, where 10 millisecond is the amount of time the dispatcher will allow for the execution of lower rate tasks. The detection time can be represented by:

$$DetTime \;=\; R4FrmSize \times [[(\frac{LDet}{LxTime}) - Tin] \bmod 8] \qquad (4.1)$$

The quotient in Equation (4.1) marks the Rate-4 frame which the error detection task in run; the *modulo 8* term comes from the realtime cycle of FTMP (eight Rate-4 frames per Rate-1 frames).

Equation (4.1) is plotted in Figure 4-2 as detection time versus frame of insertion for different workloads; two experimental runs are also plotted. The high workload data has a Rate-4 frame size of 50 milliseconds and the low workload data a 40 millisecond frame size. In comparing the data of Figure 4-2, the experimental data corresponds closely to the computed data. The reason for the multiple data points for each insertion time is that error detection can be accelerated or delayed one Rate-4 frame. The increase in the slope as the workload increases is due the lengthening of the basic frame size hence placing the error detection a further time away from the fault insertion.

Figure 4-3 shows histograms for the fault detection time with the time of fault insertion varying between graphs. The fault location is the data path, but this data is representative of the other fault locations. The time skewing between the graphs show the lengthening of the detection time as the fault insertion time moves relative to the fixed detection time. Figure 4-4 shows histograms of fault detection time for Draper's hardware fault inserted data. The

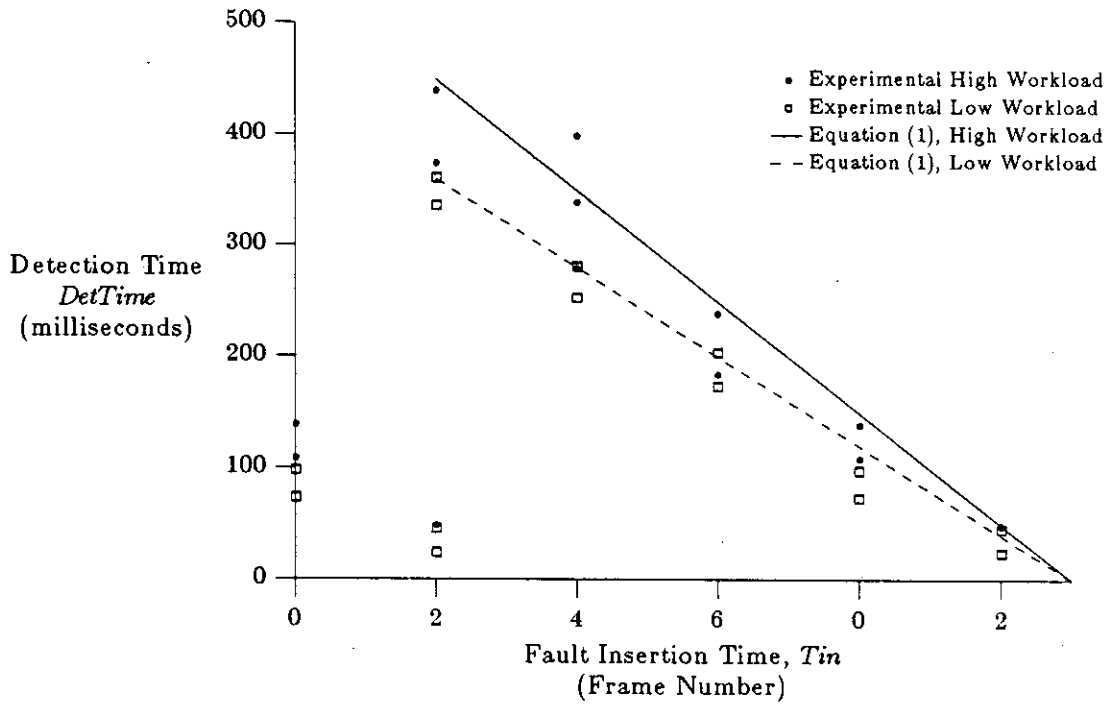

Figure 4-2: Error Detection Time as a Function of Insertion Time

detection time is approximately two times larger than the software inserted faults. The difference is due to the manifestation of faults to errors, whereas with software inserted faults the detection time is only the latency between inserting the fault and reading the error latches. Another difference between the two data sets is the distribution of the data; the software inserted faults fall into two or three groups, while the hardware inserted faults are distributed across the whole range. The random insertion time of Draper's faults and the delay in fault manifestation attribute to the continuous distribution of the data.

From the fault detection time measurements, we were able to show the parameters affecting the fault detection time. Furthermore we were able to characterize the processes from the error occurrence (error latch set) to the error detection (error latch read), but could not map from a fault occurrence to an error occurrence (Figure 2-1).

4.3 Fault Identification Time

The fault identification time is the time from the detection of an error by the system until the source of the error is identified. For both software and hardware inserted faults the expected data should be similar; the mechanisms involved are the same (Section 2.3).

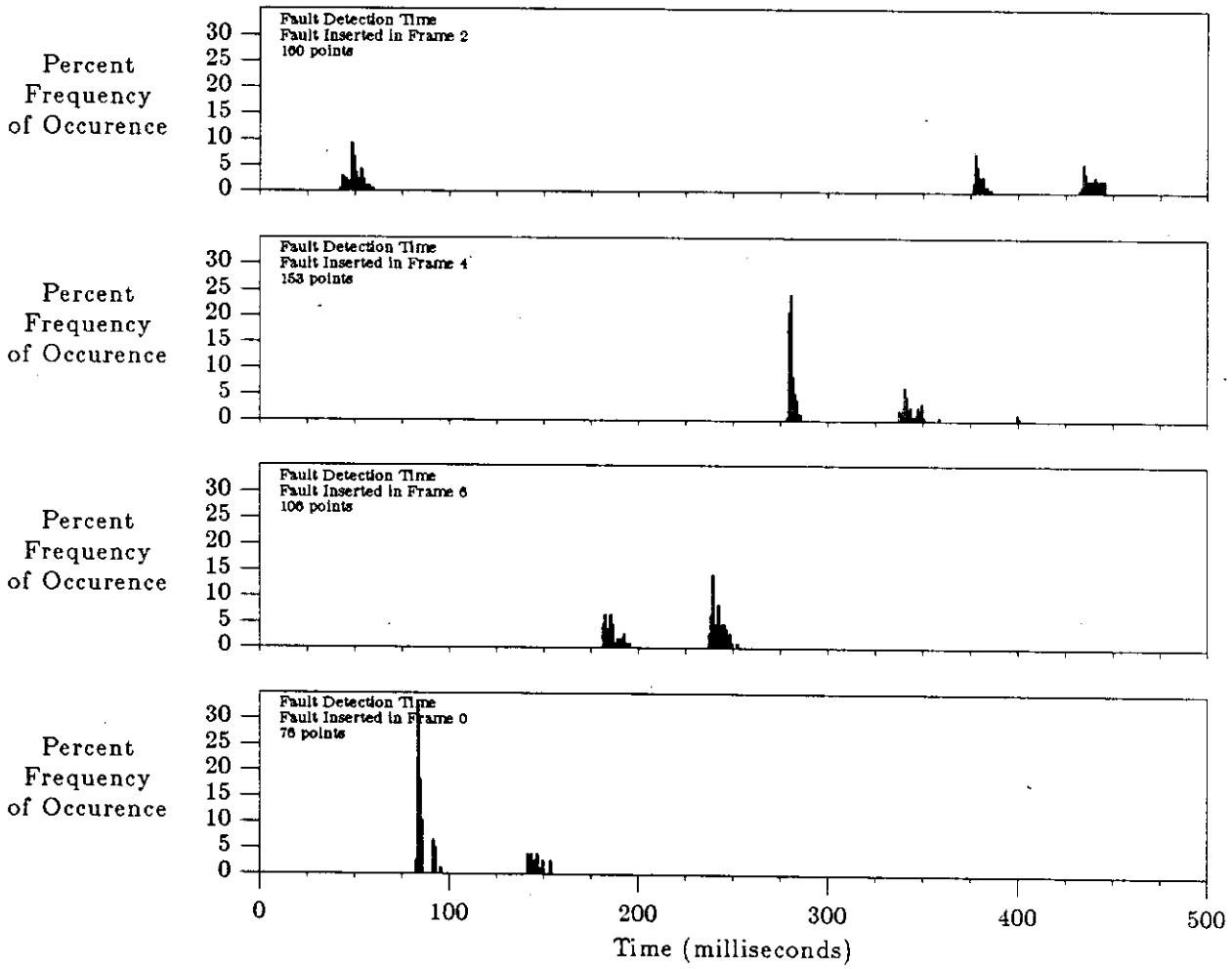

Figure 4-3: Fault Detection Time for Software Inserted Faults

The primary parameter affecting the data is the manifestation of the fault; if a fault manifests to errors on different buses then the possible sources of the fault is limited. A secondary parameter is the number of possible sources for the error. This is dependent on system configuration: the more processors, the more sources of errors. The experiments varied the fault locations (manifestation), and the system configuration (possible sources).

The data should be grouped according to the execution time of the identification routine, which is dependent on the number of suspect units. The routine runs as a Rate-1 task, once per 400 milliseconds, with the data grouped according to the number of passes.

Figure 4-5 shows histograms of fault identification times for the software inserted faults.

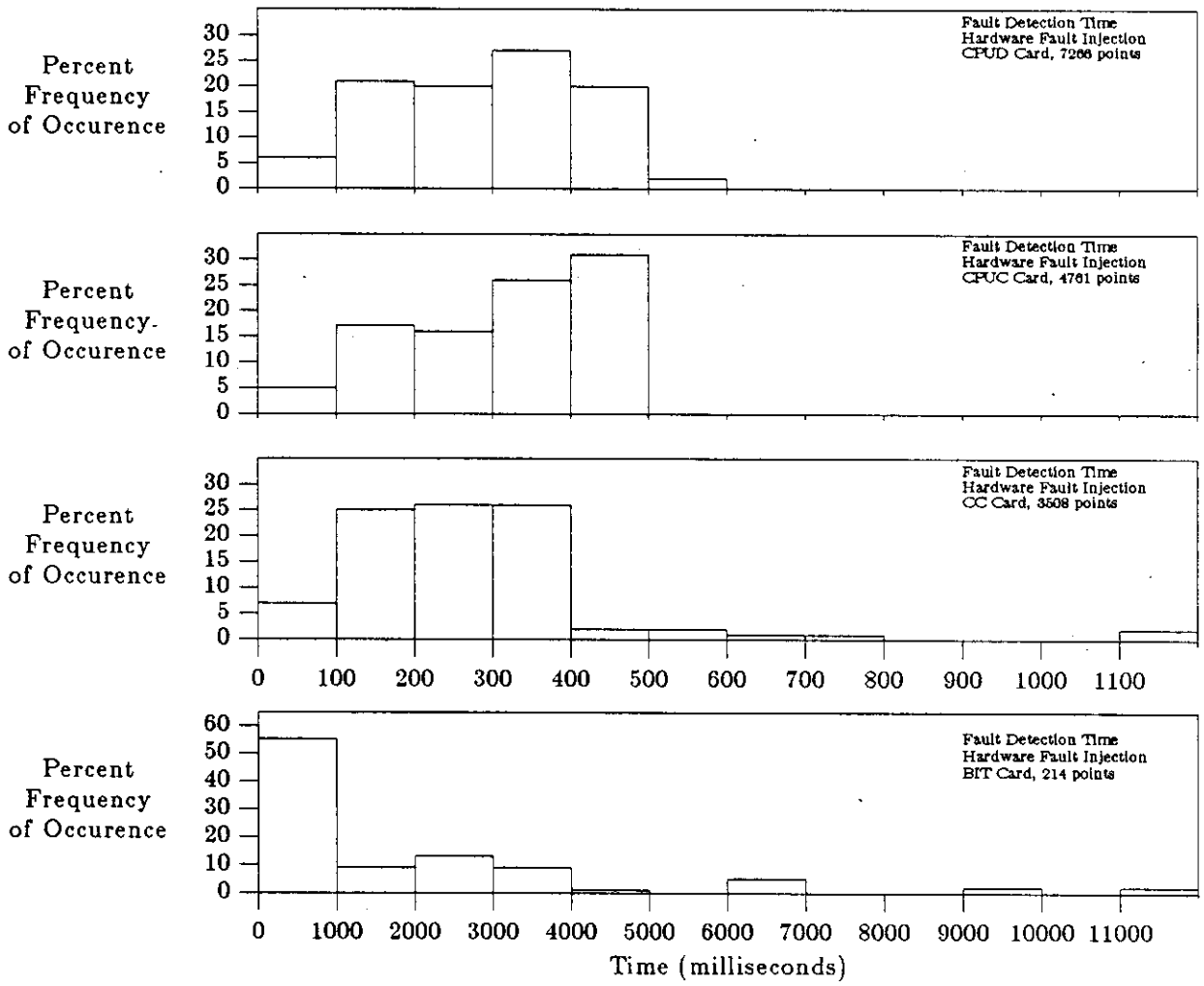

Figure 4-4: Draper's Hardware Inserted Fault Detection Time

The main feature of the data is the discrete distribution. This was expected; the distribution is in multiples of the Rate-1 frame size, approximately 400 millisecond. Thus Figure 4-5 also shows the number of execution cycles required for the identification routines.

Of interest is the control path fault with two triads executing; the identification time is under 50 milliseconds, hence the source was located without reconfiguring the system. The reason for this is as follows: the control path fault sends one processor of the triad into an infinite loop. As the other processors continue execution they will transmit on both the poll bus and the transmit bus causing errors to occur on each. These two errors are sufficient to determine the source and hence the faulty unit is identified without further information.

At the other extreme is the transmit bus fault with three triads executing. Here the number

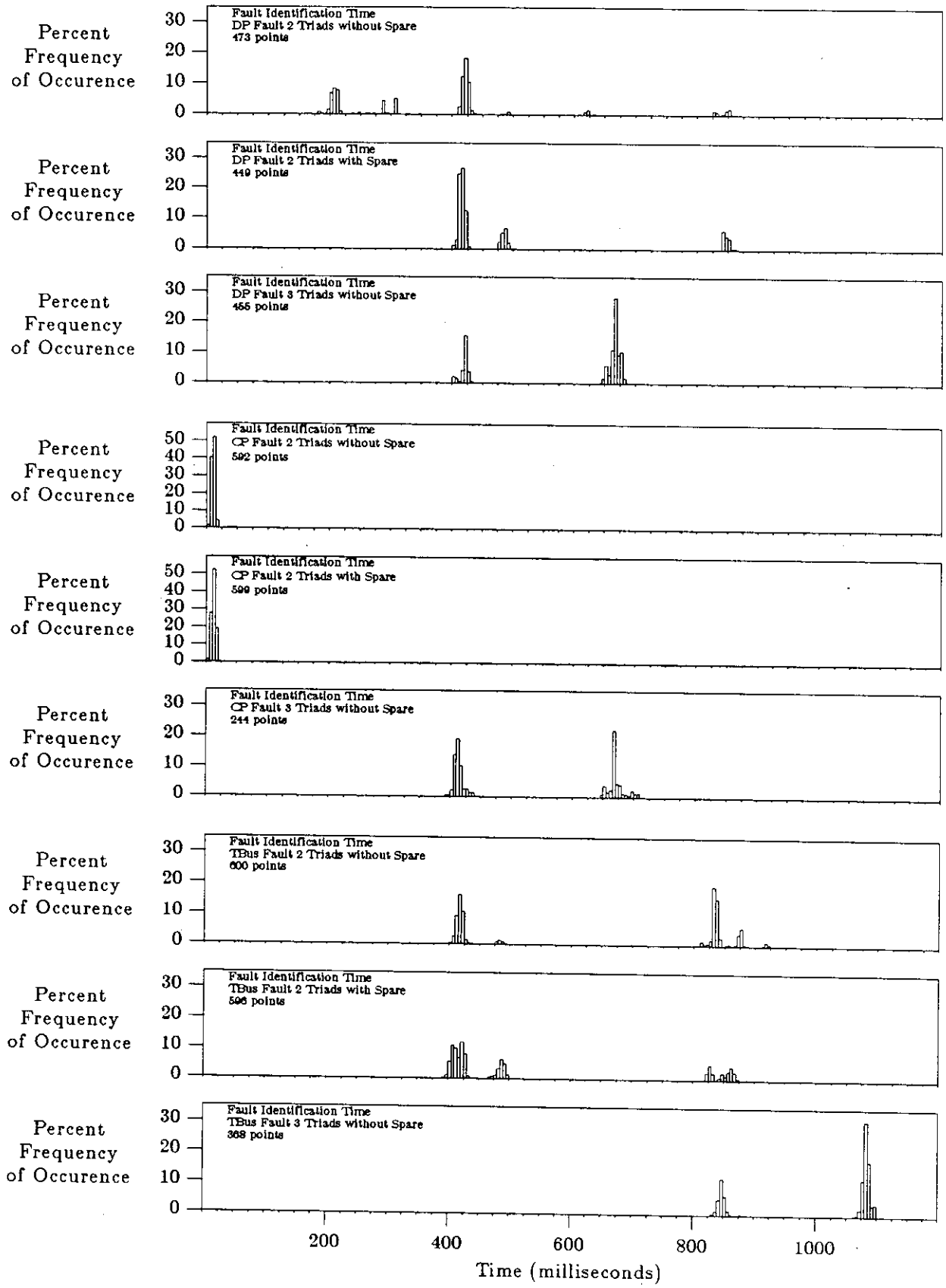

Figure 4-5: Fault Identification Time for Software Inserted Faults

of error sources is four, the bus and the three processors enabled on the bus. This should require a minimum of two bus swaps to determine the source of the error. In this example three bus swaps were required. This is due to an error in the identification routine which does not swap buses on all triads.[5]

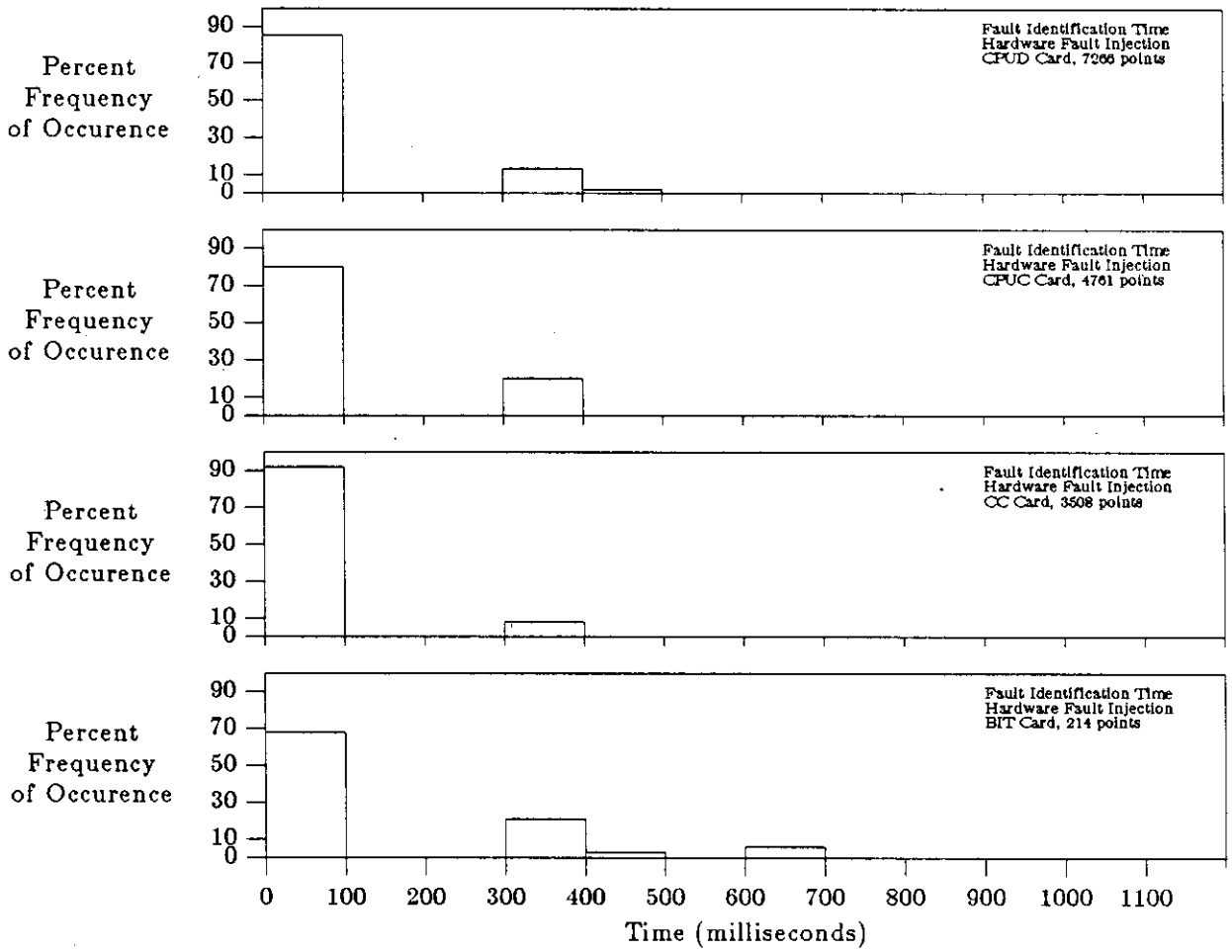

Figure 4-6: Draper's Fault Identification Time

Figure 4-6 presents histograms of Draper's data for fault identification times for the four different cards in the comparison. The major difference between Draper's data and the software inserted fault data is that a significant amount of Draper's data points lie in the first bin, 0 to 100 milliseconds, with fewer outliers at the Rate-1 frame size, 320 milliseconds. As stated earlier the error identification time is a function of the number of suspect units to which the errors can

[5]This error was further evident in the observations of the transient fault routines conducted during preliminary experiments with Software Fault Insertion.

be attributed. From analysis of Draper's data, the hardware inserted faults manifest to errors on multiple buses which can only be attributed to a single unit.

From the data, the fault identification behavior was characterized. Draper's data had mostly multiple errors, whereas the software inserted faults allowed evaluation under single detected errors. This shows software implemented fault insertion can be a tool for the evaluation and characterization of fault-handling routines.

4.4 Fault Recovery Time

The fault recovery time is the time from the identification of a faulty unit to the time when the unit is removed from the active system. The data for software inserted faults should be similar to Draper's hardware inserted faults. The primary parameter is the system configuration, the presense or absence of spares. The expected data should show an increase in recovery time when spares are not available.

Figure 4-7 shows histograms of fault recover times under various system configurations and fault locations. With the data path and control path faults, the unit failed was a processor and hence the processor was retired; for the transmit bus fault a bus was marked faulty and replace. The data shows the expected increase in recovery time when no spares are available, further more the data is grouped at 45 and 95 millisecond. This represents the period of the dispatcher which executes the reconfiguration commands.

Figure 4-8 shows Draper's fault recovery data. Their data is similar to the sum of the software inserted fault data. Draper's data lacks the resolution and specification of experimental condition for useful comparisons, but from the two data sets, it is evident software implemented fault insertion can be used to characterize and evaluate the fault recovery procedure of a system.

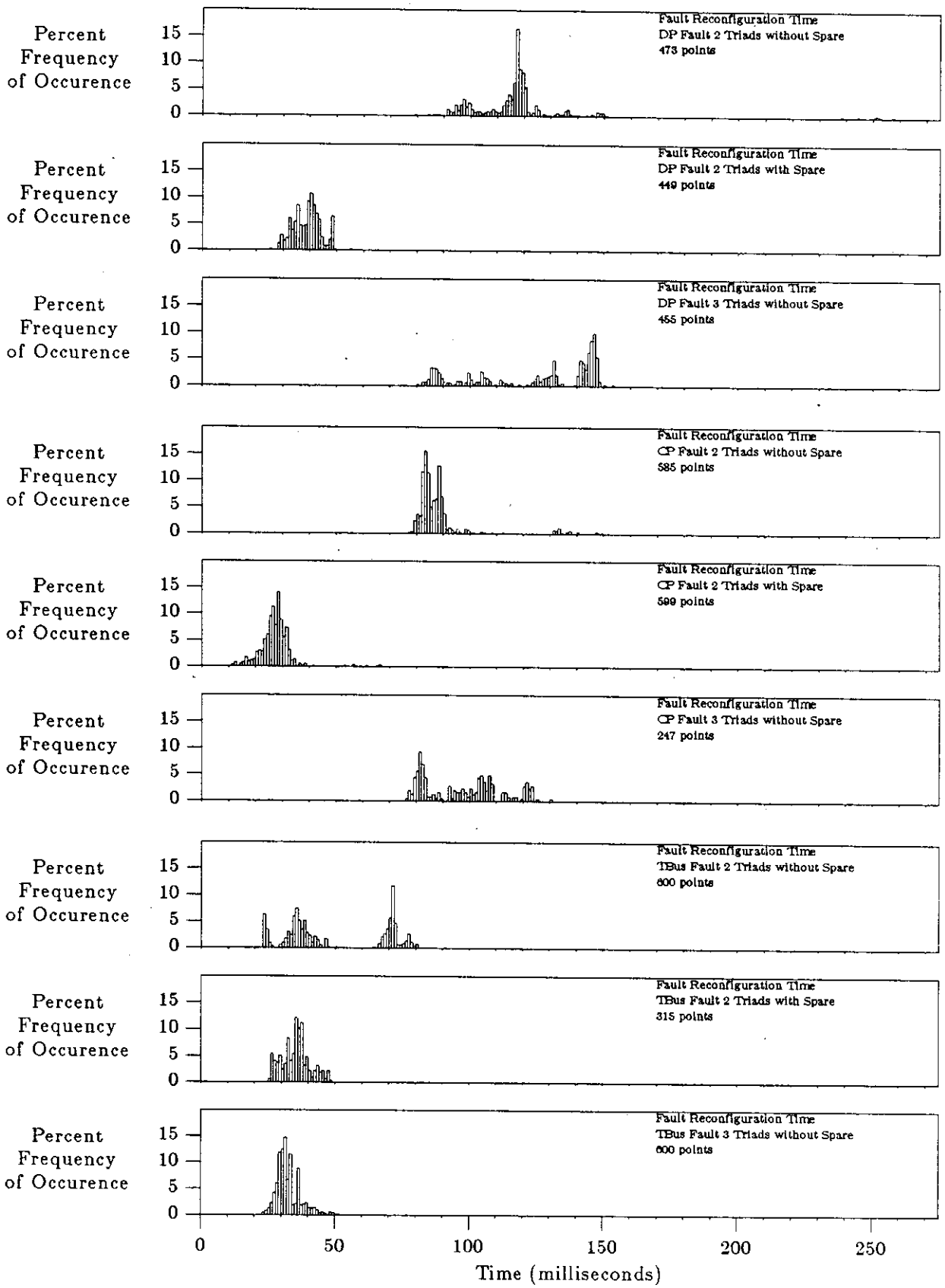

Figure 4-7: Fault Recovery Time for Software Inserted Faults

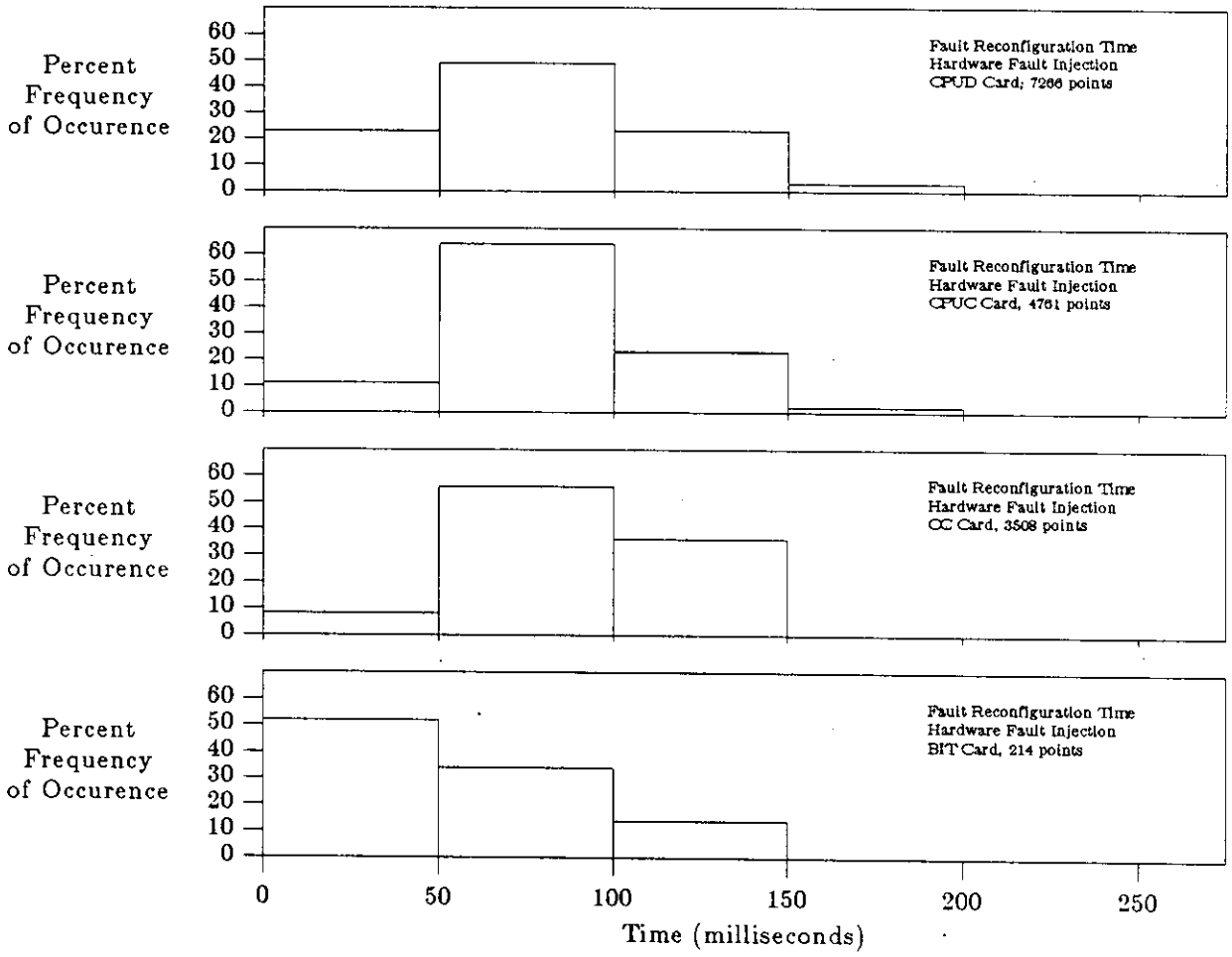

Figure 4-8: Draper's Fault Recovery Time

5. Conclusions

This paper presented a model for software implemented fault insertion and implemented the model on a fault tolerant computer, FTMP. Experiments were conducted and from the data the following information was gathered about FTMP:

- Measured Detection Times: are a function of workload, and time of insertion.
- Measured Identification Times: are a function of configuration and type of faults inserted. Errors in the identification code were uncovered by observing system response.
- Measured Recovery Times: are a function of the system configuration.

In comparison to hardware fault insertion the following points can be made regarding the two fault insertion schemes:

- Both fault insertion schemes were able to characterize the fault identification and reconfiguration times of the system.
- Hardware fault insertion places the fault at a lower level (pin level) than the software insertion (processor level). For this reason the detection times for the hardware inserted faults included the fault latency times, whereas software fault insertion only included the error detection latency, Figure 2-1.
- The fault manifestation and propagation for hardware inserted faults allows less control in the generation of specific error types than the software inserted faults. This control may be useful during the evaluating of specific fault identification routines.

In summary, although software implemented fault insertion does not fully emulate hardware fault insertion, it provides a means to evaluate the fault detection, identification, and recovery means of a system. The software fault insertion can also be used to in the characterization the systems across architectural and implementation boundaries. Furthermore, as the controllability and observability of processors decrease due to the increased used of VLSI technology, software implemented fault insertion may be a reasonable approach to system evaluation.

Software Implemented Fault Insertion: An FTMP Example

References

[Boone et al. 80] L.A. Boone, H.L. Liebergot, and R.M. Sedmak.
Availability, Reliability, and Maintainability Aspects of the Sperry UNIVAC 1100/60.
In *FTCS-10*, pages 3-9. IEEE, June, 1980.

[Draper 83a] *Development and Evaluation of a Fault-Tolerant Multiprocessor Computer, Vol. III, FTMP Test and Evaluation*
Charles Stark Draper Laboratories,.1983.
NASA Contract Report 166073.

[Draper 83b] *Development and Evaluation of a Fault-Tolerant Multiprocessor Computer, Vol. I, FTMP Principles of Operations*
Charles Stark Draper Laboratories, 1983.
NASA Contract Report 166071.

[Draper 83c] *Development and Evaluation of a Fault-Tolerant Multiprocessor Computer, Vol. II, FTMP Software*
Charles Stark Draper Laboratories, 1983.
NASA Contract Report 166072.

[Feather et al. 85]
Frank Feather, Daniel Siewiorek, and Zary Segall.
Validation of a Fault-Tolerant Multiprocessor: Baseline Experiments and Workload Implementation.
Technical Report CMU-CS-85-145, Carnegie Mellon University, July, 1985.

[Hopkins et al. 78]
A.L. Hopkins, T.B. Smith, and J.H. Lala.
FTMP - A Highly Reliable Multiprocessor.
In *Proceeding of the IEEE*, pages 1221-1237. October, 1978.

(NASA 79a) NASA-Langley Research Center.
Validation Methods for Fault-Tolerant Avionics and Control Systems - Working Group Meeting I, NASA-Langley Research Center, 1979.
NASA Conference Publication 2114.

(NASA 79b) Research Triangle Institute.
Validation Methods for Fault-Tolerant Avionics and Control Systems - Working Group Meeting II, NASA-Langley Research Center, 1979.
NASA Conference Publication 2130.

[Schuette, et al. 86]
M.A. Schuette, J.P. Shen, D.P. Siewiorek, and Y.X. Zhu.
Experimental Evaluation of Two Concurrent Error Detection Approaches.
In *FTCS-16*, pages 138-143. IEEE, July, 1986.

[Siewiorek and Swarz 82]
Daniel P. Siewiorek and Robert S. Swarz.
The Theory and Practice of Reliable System Design.
Digital Press, 1982.

[Yang et al. 85] X.Z. Yang, G. York, W.P. Birmingham, and D.P. Siewiorek.
Fault Recovery of Triplicated Software on the Intel iAPX 432.
In *Distributed Computing Systems*, pages 438-443. May, 1985.