

EXPERIMENTAL RESEARCH IN DEPENDABLE COMPUTING AT CARNEGIE MELLON UNIVERSITY

From Faults to Manifestations

Daniel P. Siewiorek

Department of Computer Science

Department of Electrical and Computer Engineering

Roy A. Maxion

Department of Computer Science

Priya Narasimhan

Department of Electrical and Computer Engineering

Carnegie Mellon University

Pittsburgh, Pennsylvania 15213

USA

1. INTRODUCTION

In 1945, the Carnegie Plan for higher education was evolved. The basic philosophy of the plan was "learning by doing". The strong emphasis on experimental research at Carnegie Mellon University (CMU) is one example of the Carnegie plan in operation. In particular, research in reliable computing at CMU has spanned five decades of researchers and students.

In the early 1960's, the Westinghouse Corporation in Pittsburgh had an active research program in the use of redundancy to enhance system reliability; William Mann, who had been associated with CMU, was one of the researchers involved in this effort. In 1962, a symposium on redundancy techniques was held in Washington, D.C., and led to the first comprehensive book [57], co-authored by Mann, on the topic of redundancy and reliability. CMU's Professor William H. Pierce wrote a paper on adaptive voting [40] that formed a part of this book. Pierce also published one of the first textbooks on redundancy [41].

During the next four decades, a large number of experimental hardware and software systems were designed, implemented, and made operational, at CMU. These systems covered a range of computer architectures, from uniprocessors, to multiprocessors, to networked systems. Each system represented a unique opportunity to include, and to quantify the results of incorporating, reliability features in the design of the system. This paper surveys the monitoring, measurement, and evaluation of those systems. A common theme has been to understand the natural occurrence of faults, to develop mathematical models for prediction, and to raise the level of abstraction of fault-models in order to monitor and design dependability mechanisms more easily. Figure 1 illustrates the breadth, diversity, depth and chronological progress of the dependability research at CMU, over the past few decades up until the present day.

2. MULTIPROCESSOR ARCHITECTURES

In 1969, Gordon Bell headed up a research seminar whose goal was to design an architecture that would be particularly suited for artificial intelligence applications. The result of the seminar was a paper study outlining *C.ai* [4]. One subunit of the *C.ai* architecture was a multiprocessor employing a crossbar switch. Ultimately, the multiprocessor portion of *C.ai* evolved into *C.mmp* (Computer.multi-mini-processor). The DARPA-funded *C.mmp* project started in 1971, became operational in mid-1975, and was decommissioned in March 1980. *C.mmp* (see Figure 2) was comprised of sixteen PDP-11 processors communicating with 16 memories through a crossbar switch *C.mmp* [58] added a minimal amount of error-detection in its hardware. The natural redundancy in its replicated processors and memory provided opportunities for substantial software error-detection and reconfiguration techniques [25].

	1970's	1980's	1990's	2000's
Monitoring	Crash Dumps (1975)	Error Logs (1980)	Natural Workloads (1990)	Distributed, Asynchronous
Fault Model	Gate Level	Register Transfer	Design, User Errors, Reactive	Attacks, Proactive (2004)
Fault Injection	Stuck-At	Memory Level (1985)	API-Level (1995)	Security (2000), Resource Exhaustion
Abstractions	Stuck-At	Error Logs \Rightarrow Clustering \Rightarrow Space/DFT (1986)	Gate \Rightarrow RT, Message \Rightarrow Fault feature vector, Memory \Rightarrow Crash (1995)	Multi-Dimensional
Modeling	Event, Mathematical Distribution/Parameters (1975)	Fault and Workload Interaction (1985)	Event Clustering, Trend Analysis, Prediction (1995)	Machine Learning

Figure 1. The depth, breadth and chronology of dependability research at Carnegie Mellon University.

In 1972, Professors Samuel Fuller and Daniel Siewiorek joined CMU. During that time, the hardware design for C.mmp was in full swing. It was a fruitful period for developing analytical models of its performance [5] and reliability [49]. The major results of the C.mmp project are described in [59].

Figure 2. The "H" shaped configuration of C.mmp with crossbar switch and memory in the center, surrounded by banks of four processors.

In the fall of 1972, a seminar was launched at CMU to explore the architectural possibilities of using microprocessors in a large-scale multiprocessor whose cost grew linearly in the number of processors employed. Architectural specifications for the computer module project were developed and reported [3], and a detailed architectural design was undertaken, culminating in the *Cm** architecture (see Figure 3) [52].



Figure 3. Two of the five clusters of Cm*. The open cluster has four computer modules on the top backplane; K.map is on the bottom.

Cm*. was extensively studied with performance and reliability models during the design process. A ten-processor system became operational in 1977. As a result, Cm*. had incorporated many more performance and reliability features than C.mmp [49], and grew into a fifty-processor experimental system, complete with two independent operating systems, that became operational in 1979. Further details on Cm*. can be found in [17].

Dan Siewiorek spent the summer of 1975 with the Research and Development group at Digital Equipment Corporation. The goal of



the summer project was to study issues of testing and reliability in computer structures. The work culminated in an architectural specification for *C.vmp* (Computer.voted multi-processor). *C.vmp* employed off-the-shelf components with little or no modification to achieve hard and transient fault survivability [51]; furthermore, *C.vmp* executed an unmodified operating system. In addition to a voting mode, the bus-level voter also allowed a non-replicated device (such as a console terminal) to broadcast results to all three processors, or it allowed the system to divide into three independent computers intercommunicating through parallel interfaces. *C.vmp* could switch between independent-mode and voting-mode operation, thereby permitting the user to dynamically trade performance for reliability [47].

C.vmp became operational in the fall of 1976, and was operational for five years. Experience indicated that *C.vmp* was about six times more reliable for transient faults than the single LSI-II systems employed in Cm* [3]. Performance degradation due to the voter was theoretically predicted and experimentally measured [45]; the voter was found to reduce the system-level performance by about 15%. The voter design was generalized to include both asynchronous and synchronous bus protocols [32][33].

At the time of *C.vmp*'s inception, engineers and designers were becoming aware of the predominance of transient errors over hard failures. A major goal of the *C.vmp* project was to use *C.vmp* as a transient meter whereby the sources of transients could be measured in much the same way that a voltmeter can measure sources and magnitudes of voltages. A statistics board was added to the *C.vmp* design [43] in order to compare the three buses for disagreements and to store the contents of all three buses (including a unique time-stamp) into a shift register when a disagreement was observed.

There exist a wide variety of applications where data availability must be continuous, that is, where the system is never taken off-line and any interruption in the accessibility of stored data causes significant disruption in the service provided by the application. Examples of such systems include on-line transaction processing systems such as airline reservation systems and automated teller networks in banking systems. In addition, there exist many applications for which a high degree of data availability is important, but continuous operation might not be required. An example is a research and development environment, where access to a centrally stored CAD system is often necessary to make progress on a design project. These applications and many others mandate both high performance and high availability from their storage subsystems.

Redundant disk-arrays are systems in which a high level of I/O performance is obtained by grouping together a large number of small disks, rather than by building one large, expensive drive. The high component-count of such systems leads to unacceptably high rates of data-loss due to component failure; thus, such systems typically incorporate redundancy to achieve fault-tolerance. This redundancy takes one of two forms: replication or encoding. In replication, the system maintains one or more duplicate copies of all data. In the encoding approach, the system maintains an error-correcting code (ECC) computed over the data. The latter category of systems is very attractive because it offers both low cost per megabyte and high data reliability; unfortunately, such systems exhibit poor performance in the presence of a disk failure. Our research at CMU addressed the problem of ECC-based redundant disk-arrays [14] that offer dramatically higher levels of performance in the presence of failure, as compared to systems comprising the current state-of-the-art, without significantly affecting the performance, cost, or reliability of these systems.

The first aspect of the problem considered the organization of data and redundant information in the disk-array. Our research demonstrated techniques for distributing the workload induced by a disk failure across a large set of disks, thereby reducing the impact of the failure recovery process on the system as a whole. Once the organization of data and redundancy had been specified, additional improvements in performance during failure recovery could be obtained through the careful design of the algorithms used to recover lost data from redundant information. The research showed that structuring the recovery algorithm so as to assign one recovery process to each disk in the array, as opposed to the traditional approach of structuring it so as to assign a process to each unit within a set of data units to be concurrently recovered, provided significant advantages.

Finally, the research developed a design for a redundant disk-array targeted at extremely high availability through extremely fast failure-recovery. This development also demonstrated the generality of the technique.

3. HARD AND TRANSIENT FAULT DISTRIBUTIONS

Often, designers must make tradeoffs between alternative reliability techniques with inadequate knowledge about how systems fail during operation. In 1977, Dan Siewiorek took a one-semester leave of absence from CMU and spent eight months working with the VAX-II /750 design team on the issues of reliability, availability, and maintainability. To answer questions about modeling hard failures, the research focused on collecting data from Cm*. Several different module types were utilized in Cm*; a chip count for each module was tabulated, followed by the modules of that type in the system, the total number of hours that these modules were utilized, and the total number of failures. The data was found to follow an exponential distribution with the failure-rate as predicted by the Military Handbook 217 [56] suitably modified to take into account the time-rate of the change of technology [49].

While substantial progress had been made in the area of understanding hard failures, transient faults posed a much harder problem. Once a hard failure had occurred, it was possible to isolate the faulty component uniquely. On the other hand, by the time a transient fault manifested itself (perhaps in the form of a system-software crash), all traces of its nature and location were long gone. Thus, research was started on the data collection and the modeling of transient faults. Data was collected from various systems – four time-sharing systems, an experimental multiprocessor, and an experimental fault-tolerant system – that ranged in size from microprocessors to large ECL mainframes. The method of detecting transient-induced errors varied widely. For the PDP-10 time-sharing systems, internally detected errors were reported in a system event-log file. For the experimental multiprocessor, Cm*, a program was written (under the guidance of Sam Fuller) that automatically loaded diagnostics into idle processors, initiated the diagnostics, and periodically queried the diagnostics as to their state. For the triply-redundant C.vmp, a manually generated crash-log was kept. Transient faults were seen to be approximately twenty times more prevalent than hard failures [34]. Gross attributes of observed transients

were recorded [49]. The data from C.mmp illustrated that the manifestation of transient faults was significantly different from the traditional permanent fault-models of stuck-at-1 and stuck-at-0¹.

4. TREND ANALYSIS

During the summer of 1978, Dan Siewiorek worked at DEC on a project whose goals were to improve the reliability, availability, and maintainability of DEC systems. The VAX cluster concept was evolving and one offshoot of the summer's activity was a diagnosis and maintenance plan for VAX clusters. Some of the advocated concepts included increased numbers of user-mode diagnostics (up until that time, the majority of DEC diagnostics executed in either stand-alone mode or under a separate diagnostic supervisor), and on-line analysis of system event-logs to determine trends and to advise the operating system of desirable reconfigurations prior to catastrophic failure. Subsequently, three separate internal DEC groups started projects in the off-line analysis of system event-logs. The extra user-mode diagnostics were used to exercise suspected system components in order to gather more evidence in the system event-log.

Back at CMU, research continued in understanding system event-logs. The first step was to analyze the inter-arrival times of transient errors. Studies of these times indicated that the probability of crashes decreased with time, *i.e.*, a decreasing failure-rate Weibull function (and not an exponential distribution) was the best fit for the data.

Because experimental data supported the decreasing failure-rate model, a natural question was "How far could you stray if you assumed an exponential function with a constant, instead of a decreasing failure-rate?" The difference in reliability – as a function of time between an exponential and a Weibull function with the same parameters – was examined. Reliability differences of up to 0.25 were found; because the reliability function can range only between 0 and 1, this error is indeed substantial [8][31]. Another area of modeling involved the relationship between the system load and the system error-rate. Software was developed to analyze system event-log entries and to statistically sample load [6][7]. From those data, a model of the system was developed which predicted failures involving hardware and software errors. Starting from first principles,² a new model – the *cyclostationary* model – was derived; this model was an excellent match to the measured data, and also exhibited the property of a decreasing failure-rate. A physical test demonstrated that, for the cost of some modeling accuracy, the Weibull function was a reasonable approximation to the cyclostationary model, with the advantage of less mathematical complexity.

A natural extension of our work with system event-logs was to analyze log entries to discover trends [55]. From a theoretical perspective, the trend analysis of event-logs was based on the common observation that a hardware module exhibits a period of (potentially) increasing unreliability before final failure. Trend analysis developed a model of normal system behavior, and watched for a shift that signifies abnormal behavior. Trend-analysis techniques based on data from normal system workloads were better suited for pointing out failure mechanisms than specification-based diagnostics are. This was because normal system workloads tended to stress systems in ways different than specification-based diagnostic programs did. Moreover, trend analysis could learn the normal behavior of individual computer installations. By discovering these behaviors and trends, it was possible to predict certain hard failures (and even discern hardware/software design-errors) prior to the occurrence of catastrophic failure.

¹ Examples included incorrect time-out indications, incorrect number of arguments pushed into or popped out of a slack, loss of interrupts, and incorrect selection of a register from a register file.

² It was assumed the system has two modes of operation: user and kernel. The probability of being in kernel mode was a random event with measurable statistics. A second random event was the occurrence of a system fault. It was assumed that the system is much more susceptible to crashing if a fault occurred while in kernel mode than if a fault occurred in user mode. Thus, a doubly stochastic process was set up between the probability of being in kernel mode and the probability of the occurrence of a system fault.

One trend-analysis method employed a data-grouping or clustering technique called *tupling* [54]. Tuples were clusters, or groups, of event-log entries exhibiting temporal or spatial patterns of features. The tuple approach was based on the observation that, because computers have mechanisms for both hardware and software detection of faults, single-error events could propagate through a system, causing multiple entries in an event-log. Tupling formed clusters of machine events whose logical grouping was based primarily on proximity and time in hardware space. A single tuple could contain from one to several hundred event-log entries.

5. AUTOMATED MONITORING AND DIAGNOSIS

The research at CMU was slowly progressing toward the online diagnosis of trends in systems. The work gained critical mass with the addition of Professor Roy Maxion in the summer of 1984. Roy had built a system at Xerox [28] wherein a network of host workstations was automatically monitored and diagnosed by a diagnostic server that employed system event-logs in making diagnostic decisions. There were three basic parts to the monitoring and diagnostic process, and correspondingly, three basic requirements for building a system to implement the process.

- **Gathering data/sensors.** Sensors must be provided to detect, store, and forward performance and error information (*e.g.*, event-log data) to a diagnostic server whose task it is to interpret the information.
- **Interpreting data/analyzers.** Once the system performance and error data have been accumulated, they must be interpreted or analyzed. This interpretation is done under the auspices of expert problem-solving modules embedded in the diagnostic server. The diagnostic server provides profiles of normal system behavior as well as hypotheses about behavior exceptions.
- **Confirming interpretation/effectors.** After the diagnostic server interprets the system performance and error information, a hypothesis must be confirmed (or denied) before issuing warning messages to users or operators. For this purpose, there must be effectors for stimulating the hypothesized condition in the system. Effectors can take the form of diagnostics or exercisers that are down-line loaded to the suspected portion of the system, and then run under special conditions to confirm the fault hypothesis or to narrow its range.

Several on-line monitoring and diagnosis projects have evolved. The first involved CMU's Andrew System, a distributed personal computing environment based upon a message-oriented operating system. The monitoring and diagnostic system for the Andrew File System, one of the first distributed, networked file systems ever developed, was a passive one because it did not actively communicate with network devices for the purposes of hypothesis confirmation or loading and running test-suites. Data collection was done with an Auto-Logging tool embedded in the operating-system kernel. When a fault was exercised, error events propagated from the lowest hardware error-detectors, through the microcode level, to the highest level of the operating system. Event-log analysis in a distributed computing environment exhibited some fundamental differences from the uniprocessor environment. The workstations, and indeed the diagnostic server, went through cycles of power-on and power-off. Yet, it was still possible to piece together a consistent view of system activity through a coordinated analysis of the individual workstations' views of the entire system. In addition, the utilization of workstation resources was highly individualized. Thus, the law of large numbers did not apply, and there was no "average" behavior, as there might have existed on a large, multi-user mainframe. The error dispersion index – the occurrence count of related error events – was developed to identify the presence of clustered error-events that might have caused permanent failure in a short period of time [23].

Data collected from the file servers over twenty-two months was analyzed; twenty-nine permanent faults were identified in the operator's log, and were shown to follow an exponential failure distribution. The error log was shown to contain events that were caused by a mixture of transient and

intermittent faults. The failure distribution of the transient faults could be characterized by the Weibull function with a decreasing error-rate, whereas that of the intermittent faults exhibited an increasing error rate. The failure distribution of the entire error-log also followed a Weibull distribution with a decreasing error-rate. The parameters of the entire error-log distribution were a function of the relationship between transient and intermittent faults, as summarized by the ratios of the shape parameters and the relative frequency of error occurrences (N_t/N_i). Simulation was used to study the mixing of the two error functions and the sensitivity of the overall parameters to varying shape parameters and N_t/N_i ratios. The simulation result was subsequently used to verify the process that was used for isolating the intermittent faults from the transient faults in the actual file-server's error-log.

It was shown that twenty-five faults were typically required for statistical techniques to estimate the Weibull parameters satisfying the Chi-Square Goodness-of-Fit Test requirements. Studying the average number of faults before repair activities showed that users would not tolerate such a large number of errors and subsequent system crashes prior to an attempted repair. Hence, the *Dispersion Frame Technique (DFT)* [24] was developed from the observation that electromechanical devices experienced a period of deteriorating performance, usually in the form of increasing error-rate, prior to catastrophic failure. DFT provided a methodology to effectively extract error-log entries (which were caused by individual intermittent faults) and a set of rules that could be used for fault-prediction. Once error-log entries associated with individual fault sources had been extracted, rules that used at most five data points were employed to predict failure. Mathematical analysis of the rules indicated that they captured the same trends deduced by the statistical analysis, supporting their validity as fault-prediction tools. Five rules were derived from the data collected on the SPICE and ANDREW networks and, in the latter case, these rules were able to predict 93% of the physical failures with recorded error-log symptoms including both electromechanical and electronic devices. The predictions ranged from one to over seven hundred hours prior to actual repair actions, with a false-alarm rate of 17%. The automatic data analysis feature is currently on-line and automatically producing warnings for maintenance personnel.

A portable version of the DFT was implemented in *Dmod* [42], an on-line system dependability measurement and prediction module. The *Dmod* architecture defines an API (Application Program Interface) between system monitoring functions and an analysis module; a second API defines the interface between user programs and the analysis module. *Dmod* continuously interprets system data to generate estimates of current and projected resource capabilities. *Dmod* incurs an overhead of less than 0.1% CPU usage and less than 0.1% memory usage in Unix systems, with about 1% memory usage in Windows NT systems.

In an Ethernet network, a common type of failure is the temporary or extended loss of bandwidth, also categorized as "soft failures" in the literature. Although the causes of soft failures vary, to the network user, such failures are perceived as noticeably degraded or anomalous performance. A second project at CMU involved a system for the active, on-line diagnostic monitoring of Andrew, the CMU campus-computing network. The monitoring system is termed "active" because it has the ability to initiate tests to confirm or deny its own hypotheses of failing network nodes or devices. The CMU/Andrew network currently supports about 600 workstations, servers, and gateways/routers. The number of nodes on the network grew from 1000 in the fall of 1986 and to over 5000 by the end of 1987. The research project, as a whole, monitored eight network routers, as well as the Computer Science Department's entire Ethernet network, for traffic and diagnostic information. Examples of monitored traffic parameters included transmitted and received packets, network load, and network collisions. Examples of monitored diagnostic parameters were CRC errors, packet-alignment errors, router-resource errors due to buffer limitations, and router-overflow errors due to throughput limitations [26].

The project used anomaly detection [29][30] as a means to signal to performance degradations that are indicative of network faults. In addition, a paradigm called the *fault feature vector* was used to describe the anomalous conditions particular to a fault. This paradigm can be used to detect instances of specific faults by determining when anomalous conditions in the network match the feature vector description. In a two-year study of the CMU Computer Science Network, the fault feature vector mechanism proved to be effective in detecting failures and in discriminating between failure types. This

mechanism was also effective at abstracting large amounts of network data to only those events that warranted operator attention; in this two-year study, over 32 million points were reduced to fewer than two hundred event-matches.

A third research project concerned fault-tolerance at the user-system interface. Roughly 30% of all system failures are due to human error [52]. Any diagnostic system eventually involves people, either to perform certain tests, or to make physical repairs or system adjustments. Rather than considering the human as a mere peripheral element, we consider the human to be a component of the overall system. This requires that care be taken to ensure fault-tolerance or fault-avoidance for human-related activities. A study was done to identify the cognitive sources of human error at the user-system interface [27]. A new interface architecture was developed to account for the cognitive limitations of users. An interface based on that architecture was built and user-tested against a pre-existing program that performed the same task, but with a traditional interface design. The interface based on the new architecture was effective in reducing user error to almost zero, and in facilitating a task speedup by a factor of four. Error reduction and task speedup are of particular importance in critical applications such as air traffic control and nuclear plant monitors.

6. TOOLS AND TECHNIQUES FOR RELIABILITY ANALYSIS

Even with the current interest in reliability, designers have had to use intuition and *ad-hoc* techniques to evaluate design trade-offs because there have been few, if any, reliability-oriented computer-aided design tools. If such a tool existed, it usually would be a stand-alone entity (*i.e.*, would not be integrated in the CAD database), and would require a reliability expert to operate it. Starting with the VAX-11/750 in 1977, through the design of the VAX 8600 and 8800, Dan Siewiorek was involved in the design and evaluation of the reliability, availability, and maintainability features of VAX CPU design and other major DEC projects. During that time, a design methodology [18] with supporting CAD tools for reliable design was developed. The experiences gained have been documented [44][45] in the literature and in a textbook.

7. EXPERIMENTAL EVALUATION OF FAULT-TOLERANT SYSTEMS

Once a fault-tolerant architecture has been designed, modeled, and built, there remains the question of whether the system meets its original specifications. Two fault-tolerant multiprocessors, FTMP and SIFT were developed and delivered to the Air-Lab facility at the NASA Langley Research Center. In 1979, a weekend workshop was held at the Research Triangle Institute to develop a suite of experiments for these fault-tolerant systems [36][37]. Starting in 1981, CMU performed a series of experiments to validate the fault-free and faulty performance of FTMP and SIFT. The methodology was developed initially from Cm*, and later transferred to FTMP. The same experiments were then conducted on SIFT to demonstrate that the methodology was robust, and that it transcended individual architectures [9][10][11][12][15].

Subsequently, the methodology has been used to assist the Federal Aviation Administration in the design of the next-generation air traffic control system. The methodology consists of a set of base-line measurements meant to characterize the performance of the system. A synthetic workload generator (SWG) was developed which allowed experimental parameters to vary at run-time. The synthetic workload generator drastically reduced the turnaround time for experimentation by eliminating the edit/compile/downlink-load portion of the experimental cycle. The avionic workload was developed, and the results of the baseline experiments were reproduced through appropriate settings of the synthetic workload generator's runtime parameters. The synthetic workload generator was modified to include software fault insertion.

8. RAISING THE LEVEL OF ABSTRACTION OF FAULTS

A number of simulators were also developed at CMU. The Instruction Set Processor (ISP) was extended to describe arbitrary digital systems from a functional level, down to and including, the gate level [1][2]. The ISP compiler and companion simulator are in use by over eighty government, university, and industrial organizations. The simulator has the ability to insert faults into memories, registers, and control logic. These faults can be permanent, intermittent, or transient. The ISP simulator and fault inserter has been used by Bendix to explore, at the gate level, the fault-tolerant properties of the SIFT computer.

Another topic of interest was generation of tests [46]. A program was written to generate functional diagnostics automatically from an ISP-like description of a digital system [21][22]. To calibrate the quality of the automatically generated functional diagnostics, a comparison was made between the manufacturer's diagnostics and the automatically generated diagnostics for a PDP-8. Using the ISP fault inserter, almost 1500 faults were inserted into the PDP-8 description, after which the respective diagnostics were simulated. The results showed that the automatically generated diagnostics had a higher detection percentage (95.57% vs. 85.5%), and required a factor-of-twenty fewer instruction executions.

The problem of assuring that the design goals were achieved required the observation and measurement of fault-behavior parameters under various input conditions. The measures included fault coverage, fault latency, and error recovery operation, while the input conditions included the applications' inputs and faults that were likely to occur during system operation. One means to characterize systems was fault injection, but the injection of internal faults was difficult due to the complexity and level of integration of contemporary VLSI implementations of the time. Other fault injection methods and models, which allowed observability and controllability of the system, regardless of system implementation, needed to be developed and validated based on the manifestation of low-level or actual faults. This research explored the effects of gate-level faults on system operation as a basis for fault-models at the program level.

A simulation model of the IBM RT PC was developed and injected with gate-level transient faults. Transient faults were selected because their behavior closely resembled actual faults occurring during system operation. To further the ties between normal operation and the simulation model, several applications or workloads were executed to observe and model the dependency on workload characteristics. Moreover, faults were injected at each cycle of the workload execution to imitate the temporary and random occurrence of transient faults.

A prediction model for fault manifestation was developed based on the instruction execution. The expected prediction coverage for the model was between 60% and 80% of all fault locations and instructions within the RT PC processor under study, thereby yielding a reduction of the fault space, by a factor up to eight, for fault-injection analysis. Prediction models of fault behavior at the system level, based on workload execution, showed that behavior was more dependent on workload structure rather than on the general instruction mix of the workload. This dependency could potentially allow applications to be modified to utilize specific error-detection mechanisms. Models for fault-injection through software were explored, with an analysis of the RT PC model showing that a subset of faults within all components was capable of being emulated by *software-implemented fault injection (SWIFI)*, with total coverage of faults in the data path. Overall, the use of SWIFI, coupled with simulation and prediction modeling, showed a factor-of-eight reduction of effort for fault-injection studies.

In summary, a general fault-prediction model based on instruction execution was developed, along with a model of fault manifestations that were injectable through software. These models coupled to reduce the fault space required during fault-injection studies. Moreover, these results aided in understanding the effects of transient gate-level faults on program behavior and allowed for the further generation and validation of new fault-models, fault-injection methods, and error-detection mechanisms.

A new fault-model for processors, based on a register-transfer-level (RTL) description, was subsequently created. This model addressed the time, cost, and accuracy limitations imposed by existing fault-injection techniques at the time. It was designed to be used with existing software-implemented

fault-injection (SWIFI) tools, but the error patterns that it generated were designed to be more representative of actual transient hardware faults than the ad-hoc patterns injected via SWIFI.

The fault-model was developed by abstracting the effects of low-level faults to the RTL level. This process attempted to be independent of processor implementation details while obtaining a high coverage and a low overhead. The coverage was the proportion of errors that were generated by gate-level faults that were successfully reproduced by the RTL fault-model. The overhead was the proportion of errors that were produced by the RTL fault-model, but not by gate-level faults. A prototype tool, *ASPHALT* [60], was developed to automate the process of generating the error patterns. As a paradigm for future fault-injection studies, the IBM RISC-Oriented Micro-Processor (ROMP) was used as a basis for several experiments. The experiments injected over 1.5 million faults and varied initial machine states, operand values, opcodes, and other parameters. Our results showed that ASPHALT was capable of producing an average coverage of over 97% with an average overhead of about 20%, as compared to a gate-level model of the ROMP. In addition, ASPHALT generated these errors over 500 times faster than the gate-level model. If additional coverage was required, methods were proposed to use a combination of RTL and gate-level simulations. If less coverage was allowed, the RTL fault-model could be abbreviated, reducing overhead further.

Traditional functional test methods were inadequate for pipelined computer implementations. We studied a design error-log, with the resulting conclusion that errors associated with pipelining were not negligible. A new methodology was developed based upon a systematic approach to generate functional test programs for pipelined computer implementations. Using information about the pipelined implementation and the target computer architecture definitions, this new methodology generated *pipeline functional test (PFT)* programs specifically designed to detect errors associated with pipeline design. These PFT programs were in machine assembly language form, and could be executed by the pipelined implementation.

Our research progressed to develop an instruction execution trace model for pipelined implementations. This model could determine the correctness of the concurrently executed instructions in a pipelined implementation. The basis for PFT program generation, the dependency graph, was a three-stage pipelined implementation of a von Neumann style computer architecture that was used to illustrate how to construct a dependency graph. A quantitative analysis approach for the number of dependency arcs exercised by a given instruction stream was presented. The determination of the total number of dependency arcs for a dependency graph was performed both for the example computer architecture and a three-stage pipelined implementation. Techniques were investigated to reduce the complexity of the analysis. A methodology for generating PFT programs, consisting of PFT modules, from the dependency graph for a pipelined computer implementation was developed. Each PFT module exercised some dependency arc in the dependency graph and automatically checked the correctness of the execution. .

Application of the PFT methodology to two different pipelined implementations of a standard computer architecture, the MIL-STD-1750A, was described. Automatic tools, using an instruction database that conveyed computer architecture information, were developed to generate the PFT programs.

The dependency-arc coverage analysis was applied to the standard architectural verification program of the MIL-STD-1750A architecture and the PFT programs. The results showed that the PFT programs covered more than 60% of all dependency arcs for the two pipelined implementations. Coverage of the standard architectural verification program was less than 1% for both implementations. Furthermore, the total length of the PFT programs was about half that of the standard architectural verification program. Finally, the PFT reported any violation of the dependency arc automatically, while the standard architectural verification program was not designed to check specifically for dependency violations. Actual design errors uncovered by the PFT programs have been described.

Research on automatic diagnosis adopted one of two philosophies. The engineering school of thought developed practical diagnostic systems that are usually specific to a device. The diagnostic theory school strived to understand the concept of diagnosis by developing general engines that diagnosed classes of simple devices.

We developed the design of practical mechanisms to diagnose a class of complex devices. The theory work developed a vocabulary of general mechanisms. This is extended to *x-diagnosis*. The engineering school determined that x-diagnosis, if feasible, would be useful for the test and repair of circuit boards in computer manufacturing. A number of issues needed to be addressed. The first difficulty was modeling – a device representation needed to be developed to reflect the high generality and high complexity of x-diagnosis. For example, the representation needed to account for dynamic behavior and feedback. To be practical, device models had to be acquired efficiently. It was shown that circuit simulation models developed by designers could be automatically transformed to device models that were useful for diagnosis.

Next, a task representation had to be developed for diagnosis. The questions to be answered included: In the face of dynamic behavior, feedback and incomplete device-models how should fault localization be done? For instance, fault-localization techniques were usually based on a fault-model. If a fault occurred that was outside this class, the technique would err. The number of different faults that a processor board could manifest was very large, making fault modeling a serious issue. Optimization was another concern. Traditionally, there had been much interest in minimizing the cost of diagnostic procedures. Was this practical for x-diagnosis? Finally, fault localization began with a known symptom, *i.e.*, a measurement at a point that was different from its expected value. For complex devices, this might not be a valid assumption. More likely was the discovery of symptoms that aggregate information about many points on the device. For example, test programs that exercised circuit boards reported error messages that aggregated information about the signal values and presented it in a concise form. How should we perform diagnosis when dealing with aggregate symptoms? Diagnostic techniques that addressed these issues were presented. The new results were used to sustain several theses:

1. Model-based diagnosis - which had been successfully used for gate-level circuits - could be scaled up to system-level circuits.
2. X-diagnosis for computer manufacturing was feasible. Specifically, the automation of test program execution - a methodology to test circuits in which diagnosis is currently done manually - was feasible.
3. A framework for researching and building x-diagnostic systems was presented.

Besides the theoretical development, the results were validated by an implementation that diagnosed digital circuits whose Verilog simulation models had 10-600 Kbytes of information. These circuits included likenesses of the VAX 8800 datapath, an Intel 386-based workstation board, *etc.* Furthermore, the techniques were shown to be applicable to devices other than digital circuits, provided that they were diagnosed in a manufacturing environment.

9. ROBUSTNESS TESTING

Benchmarks exist for performance measuring, software robustness testing, and security probing. Benchmarks are one way of evaluating COTS (Commercial Off The Shelf) components as well as monitoring network end-to-end capabilities. Synthetic workloads can also be used to inject workload during exercises to see how doctrines work when network resources are strained.

COTS (Commercial Off The Shelf) and legacy software components may be used in a system design in order to reduce development time and cost. However, using these components may result in a less dependable mission-critical system because of problems with exception handling. COTS software is typically tested only for correct functionality, not for graceful handling of exceptions, yet some studies indicate that more than half of software defects and system failures may be attributed to problems with exception handling. Even mission-critical legacy software may not be robust to exceptions that were not expected to occur in the original application (this was a primary cause of the loss of the Ariane 5 rocket initial flight in June 1996).

Because a primary objective of using COTS or legacy software is cost savings, any method of improving robustness must be automated to be cost-effective. Furthermore, machine-parseable software specifications are unlikely to be available in most cases, and for COTS software, it is possible that source code will be unavailable as well. While it can be hoped that a COTS vendor will fix any bugs reported by

users, the small number of copies purchased by a defense developer might not grant enough leverage to ensure that this is the case. Finally, new versions of COTS software may be continually released, causing quick obsolescence of any manually created software modifications.

The success of many products depends on the robustness of not only the product software, but also operating systems and third party component libraries. But, until the inception of *Ballista* [13], there was no way to quantitatively measure robustness. *Ballista* provided a simple, repeatable way to directly measure software robustness without requiring source code or behavioral specifications. As a result, product developers could use robustness metrics to compare off-the-shelf software components, and component developers could measure their effectiveness at exception handling.

The *Ballista* automated robustness testing methodology characterizes the exception handling effectiveness of software modules. For example, *Ballista* testing can find ways to make operating systems crash in response to exceptional parameters used for system calls, and can find ways to make other software packages suffer abnormal termination instead of gracefully returning error indications. The *Ballista* testing server enables users to test the robustness of their own and third-party software via the Internet. *Ballista* is a "black box" software-testing tool, and works well on testing the APIs of COTS software.

When a software module is tested with many different inputs, the response of that module typically falls into a number of "response regions". For example, for many inputs the module will work correctly, but for some other related input sets it may crash or hang. In one set of experiments, *Ballista* results [20] for fifteen operating systems tested indicated that the large majority of robustness failures were caused by single parameter values, independent of other parameter values. This means that a hardening wrapper could test just one parameter value, instead of many combinations of parameters, and still achieve significant hardening, yielding significant performance benefits. As an example, checking for null pointers in an operating system interface could reduce the measured robustness failure-rate by a third.

By associating test cases with data types rather than with module functions, specific test cases could be created and applied to newly submitted modules with minimal effort. For example, once test cases for a "pointer" data type were created, re-using the object-oriented pointer test case library could test any module that takes a pointer as an input. The net result was that the amount of work required to create test cases was sub-linear with the number of functions being tested. For example, 233 operating system functions were successfully tested with only 20 sets of test cases (one set of approximately 10 tests for each of the 20 data types required).

Each of the fifteen different operating systems' respective robustness was measured by automatically testing up to 233 POSIX functions and system calls with exceptional parameter values. The work identified repeatable ways to crash operating systems with a single call, ways to cause task hangs within OS code, ways to cause task core dumps within OS code, failures to implement defined POSIX functionality for exceptional conditions, and false indications of successful completion in response to exceptional input parameter values. Overall, only 55% to 76% of tests performed were handled robustly, depending on the operating system being tested.

Hardening can be accomplished by first probing a software module for responses to exceptional inputs that cause "crashes" or "hangs". When these robustness bugs have been identified, a software wrapper can be automatically created to filter out dangerous inputs, thus hardening the software module.

Many classifications of attacks have been tendered, often in taxonomic form. A common basis of these taxonomies is that they have been framed from the perspective of an attacker – they organize attacks with respect to the attacker's goals, such as privilege elevation from user to root (from the well known Lincoln taxonomy). Taxonomies based on goals are attack-centric; those based on defender goals are defense-centric. Defenders need a way of determining whether or not their detectors will detect a given attack. It is suggested that a defense-centric taxonomy would suit this role more effectively than an attack-centric taxonomy. Research at CMU has led to a new, defense-centric attack taxonomy [19], based on the way that attacks manifest as anomalies in monitored sensor data. Unique manifestations, drawn from 25 attacks, were used to organize the taxonomy, which was validated through exposure to an intrusion-detection system, confirming attack detectability. The taxonomy's predictive utility was

compared against that of a well-known extant attack-centric taxonomy. The defense-centric taxonomy was shown to be a more effective predictor of a detector's ability to detect specific attacks, hence informing a defender that a given detector is competent against an entire class of attacks.

10. FAULT-TOLERANCE FOR DISTRIBUTED SYSTEMS

The research on dependability gained additional emphasis in distributed fault-tolerance with the addition of Professor Priya Narasimhan in the fall of 2001. Priya had been instrumental in helping to establish an industrial standard for dependability, *i.e.*, the Fault-Tolerant CORBA standard [38], based on her research [35] on the development of the Eternal and Immune systems that aimed to provide transparent fault-tolerance and survivability, respectively, to middleware applications. The new research at CMU focused on integrating other properties, such as real-time, into fault-tolerant systems, and on examining the trade-offs in such a composition.

The *MEAD* system, born out of lessons learned and experiences [16] with other distributed fault-tolerant systems, focuses on proactive (rather than the traditionally reactive) fault-tolerance [39], which examines pre-fault symptoms and trend analysis in distributed systems, in order to compensate for faults before they happen. Proactive fault-tolerance yields reduced jitter in the presence of certain kinds of faults, and also mitigates the adverse impact of faults in a real-time distributed system. Other novel aspects, currently being emphasized as a part of this research effort, include: (i) the resource-aware trade-offs between real-time and fault-tolerance, (ii) sanitizing various kinds of nondeterminism and heterogeneity in replicated systems, (iii) mode-driven fault-tolerance for mission-critical applications, and (iv) development-time and run-time advice on the appropriate configuration and settings for the parameters (*e.g.*, number of replicas, checkpointing frequency, fault-detection frequency) that affect the fault-tolerance of applications. The approach aims to involve the use of machine learning techniques and feedback in distributed fault-tolerant systems in order to improve (and adapt) the response of the systems and their hosted applications to resource changes, faults and other dynamic behavior.

11. CONCLUSION

Fault-tolerant computing research at CMU spans researchers and students in both the Computer Science Department and the Department of Electrical and Computer Engineering. The three axes of integration that characterize the CMU research philosophy are:

Integration of Theory and Practice. CMU research has had a strong and balanced emphasis on both the development of theoretical concepts and the application of these concepts to practical systems. Experimental approaches have been a key element in our fault-tolerance research. Currently, there is a plan to establish a laboratory for the study of machine and human diagnosis of complex systems from both theoretical and experimental perspectives. The main objective of the laboratory is the exploration, definition, and implementation of advanced systems for intelligent diagnostic problem solving.

Integration of Hardware and Software. Traditional fault-tolerant techniques have been directed primarily at hardware systems. As systems become more complex and software engineering costs increase, an approach involving synergistic integration of hardware and software techniques will be essential. Most traditional techniques can be labeled as system/structure-based. One direction of our future research is algorithm-based/behavior-based fault-tolerance.

Integration of Space/Defense and Commercial. While most early research in reliable systems was geared for space and defense applications, in recent years, we have seen fault-tolerance concepts being applied in many commercial applications. We believe that research in both sectors can – and should be – mutually benefiting. There has been substantial research effort in fault-tolerant computing at CMU, and there is strong indication that this research area will continue to grow in both scope and visibility at Carnegie Mellon University.

11. ACKNOWLEDGEMENTS

The authors would like to acknowledge the graduate students and the multiple funding sources that, through the years, have made this research possible. These funding sources include BMD, DARPA, DEC, IBM, NASA, NSF, ONR, and Xerox. We also thank Laura Forsyth for her help in preparing this paper.

12. REFERENCES

- [1] Barbacci, M., Instruction Set Processor Specifications (ISPS): The notation and its application. In *IEEE Transactions on Computers*, vol. 30, no. 1, pp. 24-40, January 1981.
- [2] Barbacci, M., Barnes, G., Cattell, R. and Siewiorek, D. P., The ISPS computer description language, Carnegie Mellon University, Department of Computer Science Technical Report, 1978.
- [3] Bell, C. G., Chen, R. C., Fuller, S. H., Grason, J., Rege, S. and Siewiorek, D. P., The architecture and applications of computer modules: A set of components for digital systems design, In *Proceedings of the COMPCON Conference*, pp. 177-180, San Francisco, CA, February 1973.
- [4] Bell, C. G. and Freeman, P., C.ai – a computer architecture of AI research, In *Fall Joint Computer Conference Proceedings*, December 1972.
- [5] Bhandarkar, D.P., *Analytic Models for Memory Interference in Multiprocess Computer Systems*, PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, September, 1972.
- [6] Castillo, X., *A Compatible Hardware/Software Reliability Prediction Model*, PhD thesis, Carnegie Mellon University, July 1981, Also Department of Computer Science Technical Report.
- [7] Castillo, X. and Siewiorek, D. P., A workload dependent software reliability prediction model, In *International Fault Tolerant Computing Symposium*, pp. 279-286, June 1982.
- [8] Castillo, X., McConnel, S. R. and Siewiorek, D. P., Derivation and calibration of a transient error reliability model, *IEEE Transactions on Computers*, vol. 31, no. 7, pp. 658-671, July 1982.
- [9] Clune, F., Segall Z. and Siewiorek, D. P., Validation of fault-free behavior of a reliable multiprocessor system, FTMP: A case study, In *International Conference on Automation and Computer Control*, San Diego, CA, June 6-8, 1984,
- [10] Clune, E., Segall, Z., and Siewiorek, D. P., Fault-free behavior of reliable multiprocessor systems: FTMP experiments in AIR-LAB, NASA Contractor Report 177967, Grant NAG 1-190, Carnegie Mellon University, August 1985.
- [11] Czeck, E. W., Feather, F. E., Grizzaffi, A. M., Finelli, G. M., Segall, Z. and Siewiorek, D. P., Fault-free performance validation of avionic multiprocessors, In *Digital Avionic Systems Conference*, October 1986, Dallas, Texas.
- [12] Czeck, E. W., Siewiorek, D. P. and Segall, Z., Validation of a fault-tolerant multiprocessor: Baseline and synthetic workload measurements, Technical Report CMU-CS-85-177, Carnegie Mellon University, November 1985.
- [13] DeVale, J., Koopman, P. J. and Guttendorf, D., The Ballista software robustness testing service, In *Testing Computer Software Conference*, June 1999.
- [14] Elkind, S. and Siewiorek, D. P., Reliability and performance of error-correcting memory and register arrays, In *IEEE Transactions on Computers*, vol. 29, no. 10, pp. 920-927, October 1980.
- [15] Feather, F., Siewiorek, D. P. and Segall, Z., Validation of a fault-tolerant multiprocessor: Baseline experiments and workload implementation, Technical Report CMU-CS-85-145, Carnegie Mellon University, July 1985.
- [16] Felber, P. and Narasimhan, P., Experiences, strategies and challenges in building fault-tolerant CORBA systems, In *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 497-511, May 2004.
- [17] Gehringer, E. F., Siewiorek, D. P. and Segall, Z., *Parallel Processing: The Cm*. Experience*, Digital Press, Bedford MA, 1987.
- [18] Guise, D., Siewiorek, D. P. and Birmingham, W. P., Demeter: A design methodology and environment, In *IEEE International Conference on Computer Design/VLSI in Computers*, October 1983.
- [19] Killourhy, K. S., Maxion, R. A. and Tan, K. M. C., A defense-centric taxonomy based on attack manifestations, In *International Conference on Dependable Systems & Networks*, Florence, Italy, June-July 2004.
- [20] Koopman, P. J. and DeVale, J., Comparing the robustness of POSIX operating systems, In *International Conference on Dependable Systems & Networks*, Madison, WI, June 1999, pp. 30-37.
- [21] Lai, K. W., *Functional Testing of Digital Systems*, PhD thesis, Carnegie Mellon University, Department of Computer Science, 1981.

- [22]Lai, K. W. and Siewiorek, D. P., Functional testing of digital systems, In *Design Automation Conference*, Miami Beach, FL, June 1983.
- [23]Lin, T.-T. Y. and Siewiorek, D. P., Architectural issues for on-line diagnostics in a distributed environment, In *IEEE International Conference on Computer Design*, Port Chester, NY, October 1986.
- [24]Lin, T.-T. Y. and Siewiorek, D. P., Error log analysis statistical modeling and heuristic trend analysis, *IEEE Transactions on Reliability*, vol.39, no.4, p.419-32, October 1990.
- [25]Mashburn, H. H., The C.mmp/Hydra project: An architectural overview, In Siewiorek. D. P., Bell, C. G. and Newell, A., *Computer Structures: Principles and Examples*, pp. 350-370, McGraw-Hill. New York 1982.
- [26]Maxion, R. A., Distributed diagnostic performance reporting and analysis. In *IEEE International Conference on Computer Design*. Port Chester NY, October 1986.
- [27]Maxion, R. A., Toward fault-tolerant user interfaces, In *IFAC International Conference on Achieving Safe Real-Time Computing Systems (SAFECOMP-86)*, Sarlat, France, October 1986.
- [28]Maxion, R. A., Human and machine diagnosis of computer hardware faults. *IEEE Computer Society Workshop on Reliability of Local Area Networks*, South Padre Island, Texas, February 1982.
- [29]Maxion, Roy A. and Tan, K. M. C., Anomaly detection in embedded systems, *IEEE Transactions on Computers*, vol. 51, no. 2, pp. 108-120, February 2002.
- [30]Maxion, Roy A. and Tan, K. M. C., Benchmarking anomaly-based detection systems." In *International Conference on Dependable Systems and Networks*, pp. 623-630, New York, NY, June 2000.
- [31]McConnel, S.R., *Analysis and Modeling of Transient Errors in Digital Computers*, PhD thesis, Carnegie Mellon University, Department of Electrical Engineering, June 1981, also Department of Computer Science Technical Report.
- [32]McConnel S.R. and Siewiorek, D. P., The CMU voter chip, Technical Report, Carnegie Mellon University, Department of Computer Science, 1980.
- [33]McConnel. S.R. and Siewiorek, D. P., Synchronization and voting. In *IEEE Transactions on Computers*, vol. 30, no. 2, pp.161-164, February 1981.
- [34]McConnel. S.R., Siewiorek D. P. and Tsao, M. M., Transient error data analysis, Technical Report, Carnegie Mellon University. Department of Computer Science. May 1979.
- [35]Narasimhan, P., *Transparent Fault-Tolerance for CORBA*, PhD thesis, Department of Electrical and Computer Engineering, University of California, Santa Barbara, December 1999.
- [36]NASA-Langley Research Center, *Validation Methods for Fault-Tolerant Avionics and Control Systems - Working Group Meeting I*, NASA Conference Publication 2114, Research Triangle Institute, 1979.
- [37]NASA-Langley Research Center, *Validation Methods for Fault-Tolerant Avionics and Control Systems - Working Group Meeting II*, NASA Conference Publication 2130, Research Triangle Institute, 1979.
- [38]Object Management Group, *Fault-Tolerant CORBA Standard (Final Adopted Specification)*, OMG Technical Committee Document formal/01-12-29, December 2001.
- [39]Pertet, S. and Narasimhan, P., Proactive recovery for distributed CORBA applications, *IEEE Conference on Dependable Systems and Networks*, Florence, Italy, June 2004.
- [40]Pierce, W.H., Adaptive vote-takers improve the use of redundancy, In Wilcox R.H. and W.C. Mann, *Redundancy Techniques for Computing Systems*. pp. 229-250, Spartan Books, Washington. D.C. 1962.
- [41]Pierce, W.H., *Failure Tolerant Design*, Academic Press, New York 1965.
- [42]Shi, Y., A portable, self hosting system dependability measurement and prediction module, Electrical and Computer Engineering Department, Carnegie Mellon University.
- [43]Shombert. L., *The C.vmp Statistics Board Experiment*, Master's thesis, Carnegie Mellon University, Department of Electrical Engineering, 1981.
- [44]Siewiorek, D.P., Architecture of fault-tolerant computers, In *IEEE Computer*, vol. 17, no. 8, pp. 9-18, August 1984.
- [45]Siewiorek, D. P., Architecture of fault-tolerant computers, In D.K. Pradhan, *Fault-Tolerant Computing: Theory and Techniques*, Vol. II, pp. 417-466, Prentice-Hall, Englewood Cliffs, N.J., 1986.
- [46]Siewiorek, D.P. and Lai, K. W., Testing of digital systems, In *Proceedings of the IEEE*, pp. 1321-1333, October 1981.
- [47]Siewiorek, D. P. and McConnel, S. R., C.vmp: The implementation, performance, and reliability of a fault-tolerant multiprocessor, In *Third US-Japan Computer Conference*, October 1978.
- [48]Siewiorek, D.P. and Swarz, R. S., *The Theory and Practice of Reliable System Design*, Digital Press, Bedford MA, 1982.

- [49] Siewiorek, D. P., Kini, V., Mashburn, H., McConnel, S. and Tsao, M., A case study of C.mmp, Cm*, and C.vmp, Part I: Experiences with fault tolerance in multiprocessor systems, In *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1178-1199, October 1978.
- [50] Siewiorek, D. P., Kini, V., Mashburn, H. and Joobbani, R., A case study of C.mmp, Cm*, and C.vmp, Part II: Predicting and calibrating reliability of multiprocessor systems, In *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1200-1220, October 1978.
- [51] Siewiorek, D. P., Canepa, M. and Clark, S., C.vmp: The architecture and implementation of a fault-tolerant multiprocessor, In *International Symposium on Fault-Tolerant Computing*, Los Angeles CA, pp. 37-43, June 1977.
- [52] Swan, R. J., Fuller, S. H., Siewiorek, D. P., Cm*: A modular, multi-microprocessor. In *AFIPS: Proceedings of the National Computer Conference*, pp. 637-644, June 1977.
- [53] Toy, W. N., Fault-tolerant design of local ESS processors, In *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1126-1145, October 1978.
- [54] Tsao, M.M., *Transient Error and Fault Prediction*, PhD thesis, Carnegie Mellon University, Department of Electrical Engineering, January 1981.
- [55] Tsao, M. M. and Siewiorek, D. P., Trend analysis on system error files, In *International Fault Tolerant Computing Symposium*, pp. 116-119, June 1983, Milan, Italy.
- [56] U.S. Department of Defense, *Military Standardization Handbook: Reliability Prediction of Electronic Equipment*, MIL-STD-HDBK-217B, Notice 1, 1976.
- [57] Wilcox, R.C. and W.C. Mann, *Redundancy Techniques for Computer Systems*, Spartan Books, Washington, D.C., 1962,
- [58] Wulf, W.A. and Bell, C.G., C.mmp: A multi-mini-processor, In *AFIPS Conference*, vol. 41, pp. 765-777, Montvale, NJ, 1972.
- [59] Wulf, W.A., Levin, R. and Harbison, S., *Hydra/C.mmp: An Experimental Computer System*, McGraw-Hill. New York, New York, 1980.
- [60] Yount, C. and Siewiorek, D. P., The automatic generation of instruction-level error manifestations of hardware faults, In *IEEE Transactions on Computers*, vol. 45, no. 8, pp. 881-891, August 1996.