Architecture of Fault-Tolerant Computers: An Historical Perspective

DANIEL P. SIEWIOREK, FELLOW, IEEE

Invited Paper

Over the past 40 years computing systems have experienced over three orders of magnitude improvement in average time to failure and over six orders of magnitude improvement in work accomplished between outages. This paper surveys the approaches and techniques used to improve system reliability. A three-dimensional design space for comparing fault-tolerant systems is proposed and populated by over two dozen actual systems. A detailed description is provided for a dozen systems evenly distributed throughout the design space. The paper concludes by observing trends in the design space and projecting future developments.

I. Introduction

System reliability has been a major concern since the dawn of the electronic digital computer age. Designers of first generation computers had to overcome the inherent unreliability of the basic components which consisted of vacuum tubes and relays. Commencing with the second generation, the basic component typified by the solid-state transistor represented a dramatic improvement over first generation components. As the scale of integration increased from small/medium to large and to today's very large scale, the reliability per basic function has continued its dramatic improvement. However, today's fifth generation computers face new challenges. Due to the demand for enhanced functionality, the complexity of contemporary computers, measured in terms of basic functions, rose almost as fast as the improvement in the reliability of the basic component. Secondly, our dependence on computing systems has grown so great that it becomes impossible to return to less sophisticated mechanisms. Previously, reliable computing has been limited to military, industrial, aerospace, and communications applications in which the consequence of computer failure had significant economic impact and/or loss of life. Today even commercial applications require high reliability as we move towards a cashless/automated life-style.

This paper surveys the rapid evolution of fault-tolerant computers over the last 40 years. Table 1 summarizes these advances by comparing a first generation system, the UNIVAC I, commissioned in 1951 with data selected from Tandem's fifth-generation installed base in 1987. As we can see, at least three orders of magnitude improvement in such reliability metrics as unavailability and mean time to failure, while work accomplished between outage improvements of six orders of magnitude have been demonstrated. The following sections will trace the evolutionary development leading to these improvements. Section II motivates the increased interest in reliability. The anatomy of a failure, including a time line which illustrates commonly used reliability-related terms, is the subject of Section III. Section IV identifies and quantifies the sources of outage in contemporary computer systems. Three major dimensions (applications, techniques, and architecture) form the axes of a three-dimensional design space as described in Section V. A brief description of a dozen architectures which cover the main points in this design space are presented in Section VI. Finally, Section VII presents an historical perspective of the evolution of designs through the design space. Trends are indicated for trajectories predicted as potential evolutionary paths for future fault-tolerant systems.

Manuscript received March 19, 1990; revised July 8, 1991. This paper is based partially on [1, ch. 3] and on [1, pt. II, Introduction].

The author is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213-3890.

IEEE Log Number 9104573.

II. Why Reliability

Reliability is of critical importance in situations where a computer malfunction could have catastrophic results. Examples include the space shuttle, aircraft flight control systems, hospital patient monitors, power systems control, and electronic funds transfer. Reliability techniques have become increasingly more important for general purpose applications due to several trends.

- Harsher environments: With the advent of microprocessors, computer systems have left the benign environment of computer rooms and moved into industrial environments. The cooling air contains more particulate matter. Temperature and humidity vary widely and are frequently subject to spontaneous changes. The primary power supply may fluctuate and there may be more electromagnetic interference.
- Novice operators: As computers proliferate, the typical user is less sophisticated about the operation of the

0018-9219/91/$01.00 © 1991 IEEE

Table 1 Comparison of First and Fifth (Component) Generation Systems

	Univac I	Tandem	Improvement
Date of Measurement	**1951**	**1985–1987**	
Unavailability (Fraction)	0.17	$2.8 * 10^{-5}$	$6.2 * 10^{3}$
Mean Time To System Failure (hours)	66	$2.4 * 10^{5}$	$3.6 * 10^{3}$
Instructions executed per process between hard stopping	$4.7 * 10^{8}$	$2.6 * 10^{15}$	$5.4 * 10^{6}$

system. Consequently, the system has to be more robust, including toleration of inadvertent user abuse.

- Increasing repair costs: Likewise, field service engineers are less familiar with the complicated structures and larger variations of configurations available in contemporary systems. As the system's operational life decreases due to obsolescence and the reliability of individual components increases, the average field service engineer is exposed to fewer repeat situations thus increasing the time to diagnose.
- Large Systems. As systems become larger, there are more components that can fail. Since the overall failure rate is directly related to the failure rates of the individual components, fault-tolerant designs may be required to keep the overall system failure rate at an acceptable level.

III. Anatomy of a Failure

Designing a fault-tolerant system requires finding a way to prevent the logical fault that arises from a physical failure from causing an error. Figure 1 depicts the possible sources of an error. The following definitions apply [2], [3]:

- Failure: Occurs when the delivered service deviates from the specified service. Service can be viewed from several levels of abstraction: service delivered by a chip as viewed by another chip, or by the system as viewed by the user.
- Fault: Erroneous state of hardware or software resulting from failures of components, physical interference from the environment, operator error, or incorrect design.
- Error: Manifestation of a fault within a program or data structure. The error may occur some distance from the fault site.
- Permanent: Describes a failure, fault, or error that is continuous and stable. In hardware, permanent failure reflects an irreversible physical change. The word hard is used interchangeably with permanent.
- Intermittent: Describes a fault or error that is only occasionally present due to unstable hardware or varying hardware or software states (for example, as a function of load or activity).
- Transient: Describes a fault or error resulting from temporary conditions. The word soft is used interchangeably with intermittent and transient.

A fault can be caused by a physical failure, an inadequacy in the design of the system, an environmental influence, or the operator of the system. A permanent failure may lead to a permanent fault. Intermittent faults can be caused by unstable, marginally stable, or incorrect designs. Environmental conditions can lead to transient faults. All these faults can cause errors. Incorrect designs and operator mistakes can lead directly to errors.

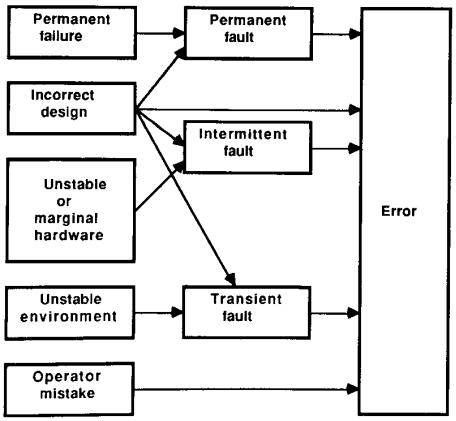

Fig. 1. Source of errors.

The manifestations of intermittent and transient faults and of incorrect hardware or software design are much more difficult to determine than permanent faults. While permanent fault models often apply to intermittents, the normal manifestations of an intermittent fault are at the system level (e.g., system crash or I/O channel retry). This occurs because most cost-sensitive contemporary small computer systems such as workstations and personal computers lack substantial on-line error detection. Furthermore, transient faults and improper designs do not have well defined manifestation. Transients arise from a combination of local phenomena (e.g., ground loops, static discharges, and power spikes) and global phenomena (e.g., alpha particles, power supply characteristics, and mechanical design).

A system may go through as many as ten stages in response to the occurrence of a failure [4]. These stages—fault confinement, fault detection, fault masking, retry, diagnosis, reconfiguration, recovery, restart, repair, and reintegration—are explained as follows. Designing a system involves the selection of a coordinated failure response that combines some or all of these steps. The ordering above corresponds roughly to the normal chronology of the stages, although the actual timing may be different in some instances.

- Fault confinement: When faults occur, it is desirable to limit the spread of fault effects to one area of the system, thereby preventing contamination of other areas. Fault confinement can be achieved through liberal use of fault-detection circuits, consistency checks before performing a function ("mutual suspicion"), and multiple request/confirmation before performing a function. These techniques may be applied in both hardware and software.

- Fault detection: Most failures eventually result in logical faults. Many techniques are available to detect faults, such as parity, consistency checking, and protocol violation. Unfortunately, these techniques cannot be perfect, and an arbitrary period of time may pass before detection occurs. This time is called fault latency. Fault-detection techniques are of two major classes: off-line detection assures integrity before and possibly at intervals during operation, but not during the entire time of operation. A typical off-line technique is the execution of a test program. On-line detection, on the other hand, provides a real-time detection capability, for it is performed concurrently with useful work. On-line techniques include parity detection and duplication.
- Fault masking: Fault-masking techniques hide the effects of failures. In a sense, the redundant information outweighs the incorrect information. In its pure form, masking provides no detection. However, many fault-masking techniques can be extended to provide on-line detection as well. Otherwise, off-line detection techniques are needed to discovered failures. Majority voting is an example of fault masking.
- Retry: In many cases a second attempt at an operation may be successful. This is particularly true of a soft fault that causes no physical damage.
- Diagnosis: If the fault detection technique does not provide information about the failure location and/or properties, a diagnostic step may be required.
- Reconfiguration: If a fault is detected and a permanent failure located, the system may be able to reconfigure its components to replace the failed component or to isolate it from the rest of the system. The component may be replaced by backup spares. Alternatively, it may simply be switched off and the system capability degraded; this process is called graceful degradation.
- Recovery: After detection and (if necessary) reconfiguration, the effects of errors must be eliminated. Normally the system operation is backed up to some point in its processing that preceded the fault detection, and operation restarts from this point. This form of recovery, often called rollback, usually entails strategies using backup files, checkpointing, and journaling. In recovery, error latency becomes an important issue because the rollback must go far enough to avoid the effects of undetected errors that occurred before the detected one.
- Restart: Recovery may not the possible if too much information is damaged by an error, or if the system is not designed for recovery. A "hot" restart (a resumption of all operations from the point of fault detection) is possible only if no damage has occurred. A "warm" restart implies that only some of the processes can be resumed without loss. A "cold" restart corresponds to a complete reload of the system, with no processes surviving.
- Repair: The component diagnosed as failed is replaced. As with detection, repair can be either on-line or off-line. In off-line repair, either the failed component is not necessary for system operation, or the entire system must be brought down to perform the repair. In on-line repair, the component may be replaced immediately by a back-up spare in a procedure equivalent to reconfiguration or the operation may continue without the component, as is the case with masking redundancy or graceful degradation. In either case of on-line repair, the failed component may be physically replaced or repaired without interrupting system operation.
- Reintegration: After the physical replacement of a component, the repaired module must be reintegrated into the system. For on-line repair, reintegration must be accomplished without interrupting system operation.

Figure 2 depicts one scenario that illustrates some of the concepts above. The time line illustrates the stages in fault handling for a nonfault-tolerance system, whereas fault-tolerant systems automatic one or more of these stages. Upon detection and the failure of retry to correct the problem, the system is brought down, diagnosed, and manually reconfigured to allow a restart. Before operation recommences, the software process must first be rolled back to a point before the errors occurred, and then restarted. Finally, after the failed module is repaired and put back on line, the system is halted temporarily to allow the module to be reintegrated into the system. Figure 2 also illustrates some of the reliability measurement concepts: mean time between failures (MTBF), mean time to detection (MTTD, sometimes called error latency), mean time to repair (MTTR), and availability (a fraction of the time system is able to produce useful results).

IV. Sources of Outage

Systems are composed of a hierarchy of levels. Faults and errors may be generated at any of the levels in the hierarchy. Indeed, mechanisms for each of the ten stages in handling a fault (confinement, detection, masking, retry, diagnosis, reconfiguration, recovery, restart, repair, and reintegration) can be proposed at each level. Figure 3 is an incomplete example of a hypothetical system composed of five hierarchical levels. Typical errors, typical techniques for the detection and recovery stages of fault handling, and typical error response times are also given. If an error is not detected at the level in which it originated, the detection of the error is left to higher levels. Likewise, if the current level lacks the capacity to recover from a particular detected error, appropriate information about the detected error must be passed on to a higher level. As an undetected error propagates up the levels in the hierarchy, it affects an increasing fraction of the system state and data structures. Longer response times to an error mean that the error manifestations have become more diverse. Consequently, the error recovery becomes more complex. If left totally to software, error recovery routines may easily become more complex than the application software.

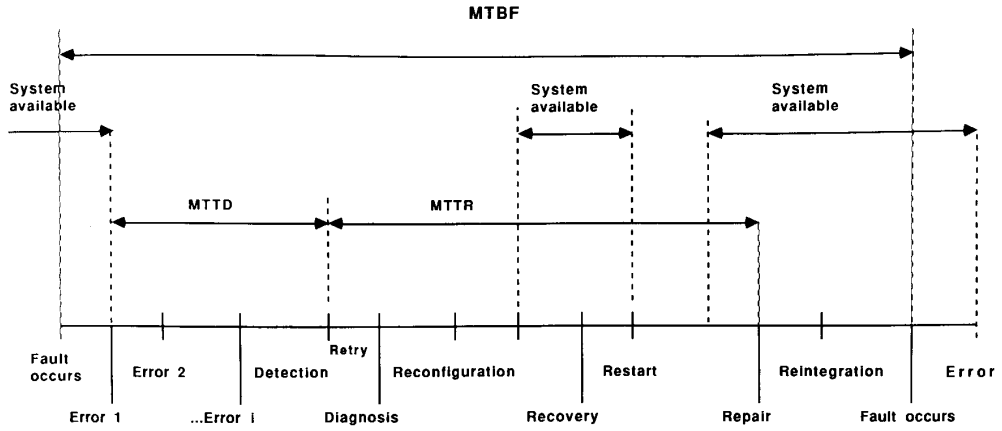

Fig. 2. Scenario for on-line detection and off-line repair due to a permanent fault. The measures MTBF, MTTD, and MTTR are the average times to failure, to detection and to repair.

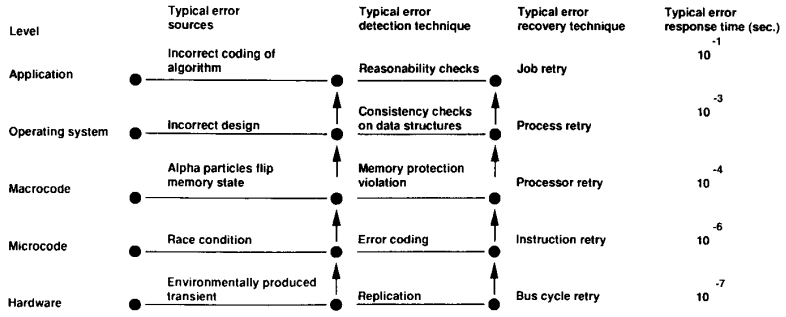

Fig. 3. Levels in a hypothetical system.

Not only are the physical levels important for describing a digital computer, but a time dimension is also required. At what point a technique or methodology is applied during the life cycle of a system may be more important than at what physical level. Table 2 lists the various stages in the life of a system as it progresses from concept to final implementation. These stages include specification of I/O relationships, logic design, prototype debugging, manufacturing, installation, and field operation. Deviations from intended behavior, or failures, can occur at any stage as a result of incomplete specifications, incorrect implementation of a specification into a logic design, and assembly mistakes during prototyping or manufacturing.

During operational life of the system, errors can result from change in the physical state or damage to hardware. Physical changes may be triggered by environmental factors such as fluctuations in temperature or power supply voltage, static discharge, and even alpha particle emissions. Inconsistent states can also be caused by operator errors and by design errors in hardware or software. Relatively, operational causes of outage are evenly distributed among hardware, software, maintenance actions, operations, and environment.

Table 2 Stages in the Development of a System

Stage	Error Sources	Error Detection Techniques
Specification and design	Algorithm design Formal specifications	Simulation Consistency checks
Prototype	Algorithm design Wiring and assembly Timing Component failure	Stimulus/response testing
Manufacture	Wiring and assembly Component failure	System testing Diagnostics
Installation	Assembly Component failure	System testing Diagnostics
Operational life	Component failure Operator errors Environmental fluctuations	Diagnostics

Design errors, whether in hardware or software, are those caused by improper translation of a concept into an operational realization. Closely tied to the human creative process, design errors are difficult to predict. Gathering statistical information about the phenomenon is difficult because each design error occurs only once per system.

Table 3 Probability of Operational Outage due to Various Sources[1]

	AT&T[2] Switching Systems [11]	Bellcore[2] [12]	Japanese Commercial Users	Tandem [13]	Tandem [13]	Northern Telecom	Mainframe Users
Hardware	0.20	0.26[4]	*[7]	0.18	0.19	.19	.45
Software	0.15	0.30[5]	0.75[7]	0.26	0.43	.19	.20
Maintenance	—	—	*[7]	0.25	0.13	—	.05
Operations	0.65[3]	0.44[6]	0.11	0.17	0.13	.33	.15
Environment	—	—	0.13	0.14	0.12	.28[8]	.15

[1]Dashes indicate that no separate value was reported for that category in the cited study.

[2]Fraction of downtime attributed to each source. Downtime is defined as any service disruption that exceeds 30 seconds duration. The Bellcore data represented a 3.5 minute downtime per year per system.

[3]Split between procedural errors (0.3) and recovery deficiencies (0.35).

[4]47% of the hardware failures occurred due to the second unit failing before the first unit could be replaced.

[5]Recovery software

[6]Split between procedural errors (0.42) and operational software (0.02).

[7]Study only reported probability of vendor-related outage (i.e., 0.75 is split between vendor hardware, software, and maintenance).

[8](.15) attributed to power.

The rapid rate of development in hardware technology constantly changes the set of design trade-offs, further complicating the study of hardware design errors. In the last fifteen years there has been some progress in the use of redundancy—using additional resources beyond the minimum required to perform the task successfully—to control software design errors [5]–[9].

Any source of error can appear at any stage; however, it is usually assumed that certain sources of error predominate at particular stages. Furthermore, error-detection techniques can be tailored to the manifestation of fault sources. Thus as each stage of system life there is a primary methodology for detecting errors.

Due to the "holes" in error detection and recovery, it is usually not possible to accurately attribute a source cause to an error. Frequently, the only error manifestations visible are gross system-level characteristics such as crash, a task stop, overdue response, etc. Table 3 depicts the relative frequency of outage attributed to five major sources across a variety of systems ranging from general-purpose commercial to dedicated fault-tolerant applications [1]. As we see from the table there is no single dominant source of outage. Equal attention must be given to the various sources if more than a factor of two improvement is sought in system reliability.

V. Fault-Tolerant Architectural Design Space

This section defines the three major axes of the space of fault-tolerant designs: system application, system structure, and fault-tolerant technique employed.

Computer systems can be taxonomized by positioning each example in a multiple-dimension design space. We will use two (function and structure) of the seven major computer space dimensions defined in [14] augmented by a third (fault-tolerant techniques) unique to fault tolerant computers. This three-dimensional framework will be used to position individual systems with respect to each other as well as to predict historical trends.

A. Applications

As in all-systems design, the system function constrained the design space, and thus the design techniques, that can be used. At the highest level of specification, fault-tolerant systems are categorized as either high availability or high reliability.

The availability of a system as a function of time, $A(t)$, is the probability that the system is operational at the instant of time, t. If the limit of this function exists as time goes to infinity, it expresses the expected fraction of time that the system is available to perform its designed function. Activities such as preventive maintenance and repair reduce the time that the system is available to perform its assigned function. Availability is typically used as a basis for evaluating systems in which functionality can be delayed or denied for short periods without serious consequences.

The reliability of a system as a function of time, $R(t)$, is the conditional probability that the system has survived the interval $[0, t]$, given that it was operational at time $t = 0$. Reliability is used to describe systems in which it is not feasible to repair (as in computers on board satellites) or in which the computer is serving a critical function and cannot be lost even for the duration of a replacement (as in flight control computers on an aircraft) or in which the repair is prohibitively expensive. In general, it is more difficult to build a highly reliable computing system than a highly available one because of more stringent requirements imposed by the reliability definition.

Frequently, characteristics of the function to which the computer system is applied can be used to improve error detection and/or recovery from failures. Four different

application types have been identified (paraphrase from [15]). The applications are ordered by increasingly stringent reliability requirements.

1) General-purpose commercial systems: High performance general-purpose computing systems are very susceptible to transient errors (due to close timing margins) and permanent faults (due to their complexity). As performance demands increase, fault tolerance may be the only course to building commercial systems with sufficient mean time to errors (MTTE) to allow useful computation. Occasional errors that disrupt computing for several seconds are tolerable as long as automatic recovery follows. Since the applications are general-purpose, there are few universal attributes which can be used to simplify the system designer's task.

2) High availability: High availability systems share resources where the occasional loss of a single user is acceptable, but a system-wide outage or common database destruction is unacceptable. These systems are most frequently oriented towards special purpose computing, executing programs whose demands can be anticipated. An example application for high availability systems is transaction processing. Transaction processing can be represented by a stylized computational model wherein all accesses to the system are in the form of a small number of requests which trigger relatively large chunks of computation (typically on the order of milliseconds) potentially resulting in a change of state reflected in primary memory and/or secondary storage. Once data has been committed to the system it cannot be lost or corrupted. However it is perfectly acceptable to require a user to resubmit a request if it could not be successfully completed due to errors.

3) Long life: Long life systems such as unmanned spacecraft cannot be manually maintained over the system operational life (frequently five or more years). Often, as in spacecraft encounters with planets, the peak computation requirement is at the end of system life. These systems are highly redundant, with enough spares to survive the mission with the required computational power. Redundancy management may be performed automatically (e.g., on board the spacecraft) or remotely (e.g., from ground stations).

4) Critical Computations: The most stringent requirement for fault tolerance is in real time control systems, where faulty computation could jeopardize human life or have high economic impact. Computations must not only be correct, but recovery time from faults must be minimized. Specially designed hardware is employed with concurrent error detection so that incorrect data never leave the faulty module.[2]

[2] [15] identified a fifth application type—postponed maintenance—which is closely related to long life systems. These are typically mobile systems that depart from a central facility for a period of time and return. The desire is to postpone maintenance until the mobile unit can return to the central facility. Systems stationed at remote sites such as weather service instruments, seismic telemetering systems, aircraft navigation beacons, and microwave relay stations are examples in which design for postponed maintenance is desirable. Since this category is subsumed by the long life category (i.e., maintenance is postponed for the entire life of the system) it will not be considered further here.

B. Structure

A notation for summarizing computer structures was defined in [14]. In the processor-memory-switch (PMS) notation, major architectural components are represented by single capital letters with P standing for processor, M for memory, S for switch, L for link, D for data operator, K for controller, T for transducer, and C for computer. Components can have modifying attributes such as P.central for central processor, which is abbreviated as Pc. Other commonly used attributes are p for primary (e.g., Mp) and s for secondary (e.g., Ms). Figure 4 depicts the canonical uniprocessor using the PMS notation. Figure 4 expands the processor into a data part D, containing arithmetic units, and a control part K containing instruction decoders and interpreters. While I/O has been left out of Fig. 4 to simplify our discussion, one should not underestimate the significance of I/O. Higher performance computer structures can be derived by replicating various combinations of M, D, and K. Replication of all three components (with the control and data components yielding a processor) leads to the multicomputer organization in Fig. 5. Each element is a complete computer with its own operating system. Computers communicate through a local area network or a high speed backplane. Any intercomputer communication must go through multiple copies of the operating system (i.e., both sender and receiver), greatly increasing the overhead of cooperation. Multicomputers have most often been used in fault-tolerant system where the physical separation of computers is believed to minimize the possibility of multiple, simultaneous hardware failures.

Replicated control and data operators that share a large main memory yields the multiprocessor configuration of Fig. 6. All processors have equal access to a shared memory, meaning that resources can be effectively shared (i.e., there needs to be only one copy of the operating system rather than one for each processor as in the multicomputer), and programs can cooperate with a minimum of overhead (i.e., processors can check flags and memory without involving the operating system).

C. Techniques

There are two approaches to increasing reliability: fault avoidance and fault tolerance. Fault avoidance results from conservative design practices such as the use of high reliability components, component burn-in, and careful signal path routing. The goal of fault avoidance is to reduce the possibilities of failure. Even with the most careful fault avoidance, however, failures will eventually occur and result in system failure (hence the name fault intolerance).

In fault tolerant designs, redundancy is used to provide the information needed to negate the effects of a failure. The redundancy is manifested in one of two ways: extra time or extra components. One form of time redundancy involves extra executions of the same calculation, perhaps by different methods. Comparisons or other operations on the multiple results (identical when no errors are present) provide the basis for subsequent action.

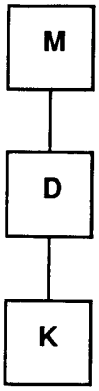

Fig. 4. Serial uniprocessor structure where M is the memory, D is a data operator, and K is a controller.

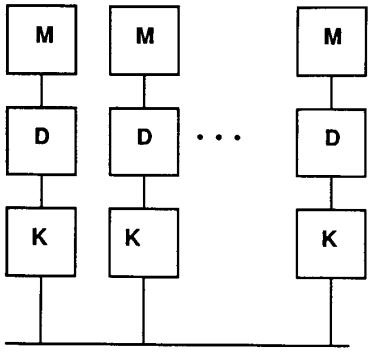

Fig. 5. A loosely-coupled multicomputer system.

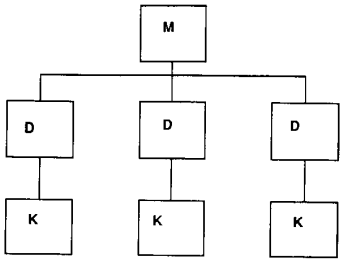

Fig. 6. A tightly-coupled multicomputer system.

Building systems that tolerate failures have been of interest since the 1940's when computational engines were constructed from relays [16]. Theoretical underpinnings of the impact of redundancy originated in the 1950's [17] as well as the formal development of the concepts of fault-tolerant computing [18], [19].

The spectrum of fault tolerant techniques can be divided into three major classes: fault detection, masking redundancy, and dynamic redundancy (see Fig. 7). A listing of typical fault-avoidance and fault-tolerant techniques are listed in Table 4. More details of these various techniques can be found in [1], [4], [6], and [20].

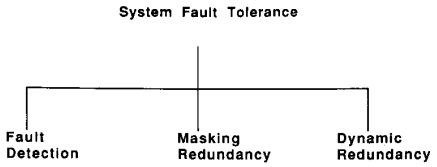

Fig. 7. Dimensions of fault-tolerant strategies.

Table 4 Classification of Reliability Techniques

Region	Technique
Fault avoidance	Environment modification
	Quality changes
	Component integration level
Fault Detection	Duplication
	Error detection codes
	M-of-N codes
	Parity
	Checksums
	Arithmetic codes
	Cyclic codes
	Self-checking and fail-safe logic
	Watch-dog timers and timeouts
	Consistency and capability checks
Masking redundancy	NMR/voting
	Error correcting codes
	Hamming SEC/DED
	Other codes
	Masking logic
	Interwoven logic
	Coded-state machines
Dynamic redundancy	Reconfiguration duplication
	Reconfigurable voting
	Backup sparing
	Graceful degradation
	Recovery

Fault detection provides no tolerance to faults, but gives warning when they occur. It is used in many small systems such as micro and minicomputers, some of which may incorporate simple on-line detection mechanisms. If the dominant form of faults is transient/intermittent, recovery can be initiated by retry invoked from a previous checkpoint in the system at whose time the system state was known to be good. These recovery methods are generally referred to as "rollbacks" because the system, upon detecting an error, rolls itself back in time to a point before the error, then begins computation again. Major error-detection techniques include duplication (frequently used for random logic) and error detecting codes (used when a group of signals representing a vector of information is changed at the same time). In duplication, two identical copies of hardware run the same computation and compare their results. When the results do not match, an error has been detected [11], [21]. Error-detection codes utilize information redundancy. For example, parity codes involve the addition of an extra bit

to represent whether the number of ones in a group of bits is odd or even. If the group of bits has an even number of ones, it is defined as having even parity; otherwise, it is odd parity.

Masking redundancy, sometimes known as static redundancy, uses extra components to obscure, or mask, the effect of a faulty component. The technique is regarded as "static" because once the redundant copies of an element are connected in the circuit, their interconnections remain fixed. The errors resulting from faulty components are "masked" by the presence of other copies of those components. Figure 8 depicts the most general masking technique known as triple modular redundancy (TMR). Three identical copies provide a voting element with a separate result. The modules may occur at any level in the system from gates through microprocessors through software through entire systems. The voter accepts outputs from all three modules and produces a majority vote at its output. This technique originated with von Neumann [17].

Error-correcting codes (ECC) are the most commonly used means of masking redundancy. One form of ECC called modified Hamming [22] single error correcting and double error detecting (SEC-DED) code is used extensively in primary memory. Because the density of memory chips is increasing, they are more prone to transient faults such as memory cell charge loss caused by external stimulation. Random access memories constitute an increasingly large part of digital systems and currently contribute as much as 60% of system failure rates. The redundancy of codes varies from 10% to 40% depending on the data bit length. Other discussion of codes can be found in [23]–[27].

The right-most branch of Fig. 7 covers those systems whose configuration can be dynamically changed in response to a fault or in which masking redundancy supplemented by on-line fault detection, allows on-line repair. Reconfiguration is triggered either by internal detection of errors in the damaged subunit or detection of errors in its output. The reconfiguration itself can occur either automatically by the system or manually by technical personnel. Error detection techniques form the basis for dynamic redundancy, and the system's chance of successful reconfiguration is largely depended upon its error detection ability.

Duplication and comparison provides error detection upon disagreement but does not provide fault tolerance. A duplicate system can become fault-tolerant provided there is a way of determining which module is faulty upon disagreement and the faulty module can be disconnected from the comparison unit. Figure 9 depicts reconfigurable duplication and only the active unit is connected to the system output. The standby unit functions in parallel with the active unit and provides the source for comparison. When a comparison detects a failure, the good unit will be switched in so that its output supplies the external hardware and the faulty unit is switched off-line. Many techniques exist for determining the faulty unit including diagnostic and fault-detection circuitry built into each unit.

AT&T switching processors use diagnostic programs and self-checking circuits [11], UDET employs self-checking

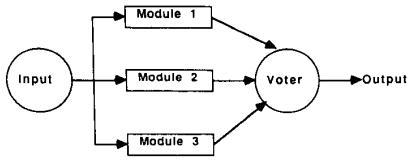

Fig. 8. Triple modular redundancy.

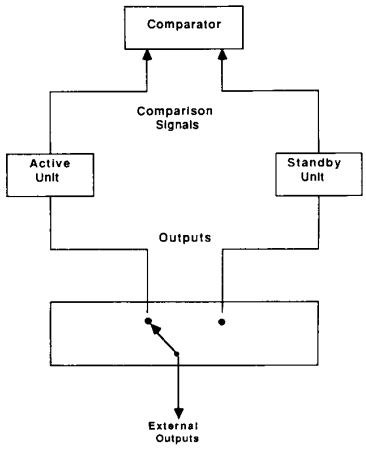

Fig. 9. Reconfigurable duplication.

[28]; the railroad traffic control computer COMTRAC uses the results of a test program compared to a precomputed result [29], and a Boeing prototype aerospace computer uses dynamically computed keywords [30].

Recovery techniques can restore enough of the system state to allow a process execution to restart without loss of acquired information. There are two basic approaches: forward and backward recovery. Forward error recovery, which produces correct results through continuation of normal processing, is usually highly application-dependent as described in [31]. Backward recovery techniques require some redundant process and state information to be recorded as computations progress. The information is used to "rollback" an interrupted process to a point for which correct information is available. Common backward recovery techniques include retry, checkpointing, and journaling. Retry techniques are the fastest form of error recovery, and conceptually the simplest. Immediately after an error is detected, the operation is retried. If the error is transient, the system will pause long enough for the transient to die away, then retry. For hard failures, reconfiguration is attempted as in mapping bad spots on disks to bad page table entries. Retries are frequently used in mainframe computers at many levels ranging from bus transactions to instructions to I/O operations [32]. In checkpointing, some subset of the system is saved at specific "checkpoints" during the process execution. Rollback consists of resetting the system to the state stored at the latest checkpoint. The only loss incurred is due to the extra time taken to perform the operation and whatever data was received

Table 5 Taxonomy of Fault-Tolerant Architectures and Examples

Application	Architecture	Technique	Example
General Purpose	Uniprocessor	Detection	**Univac I**
			IBM 3090
			DEC VAX
High Availability	Multicomputer	Standby	Sage, RW 40
			3B20D
		Dynamic	**ESS**
			Tandem
			Stratus
	Multiprocessor	Dynamic	Pluribus, Parallel
			Sequoia
			Intel 432
Critical Computations	Uniprocessor	Masking	Saturn V
			C.vmp
			FTP
	Multicomputer	Masking	Space Shuttle
			SIFT, AIPS
Long Life	Uniprocessor	Masking	OAO, **Star**
	Multicomputer	Standby	**Voyager**
			Galileo

during the interval that cannot be recreated. The Fault-Tolerant Spaceborne Computer [33]–[35], COPRA [36], [37], the JPL Self-Test and Repair (STAR) computers [38], and the Tandem computer [1], [4], [6], [20] employ checkpointing. In journaling, a copy of the initial data base is saved when the process begins. As the process executes, a record is kept of all transactions that affect the data. If the process fails, its affect can be recreated by running a copy of the backup data through the transactions a second time [39]. Many contemporary text editors use journaling to recover from computer crashes. The effectiveness of recovery techniques is a strong function of coverage—the conditional probability that if there is an error that the system successfully traverses the stages of fault detection, diagnosis, reconfiguration, and recover [40], [41].

Other dynamic redundancy techniques include reconfigurable voting, backup sparing, and graceful degradation [1], [4], [6], [20].

VI. Brief Description of Representative Architectures

The taxonomy presented in the last section yields 24 possible combinations (four categories of computer function × three computer structures × three branches of fault-tolerant techniques). However, only a subset of this design space has been populated to date. Table 5 depicts eight of the possible 24 design points and gives examples of systems representing each category. The systems identified in bold face which are representative of each major design subspace will now be briefly described to give the reader a better understanding of techniques used. To simplify the discussion we do not describe power supplies or physical packaging. All fault tolerant systems have redundant power supplies and packaging to facilitate repair.

A. General Purpose

1) Univac I: Delivered to the National Bureau of the Census in March of 1951, the Univac I was placed in service as the first commercial computer. Figure 10 represents a block diagram of the major portions of Univac I with the terminology updated to more common usage [42]. The Univac I processed words composed of 12 six-bit characters. An instruction buffer prefetched a word containing two instructions. The instruction currently under execution was used to drive a table look-up from which individual gate control signals were generated. A cycle counter was used to step through the function table and long, complex instructions required multiple passes as kept track by the stage counter. Due to the complexity and unreliability of the vacuum tube components, the engineers were particularly interested in detecting unexpected situations. Table 6 summarizes the hardware error-detectors built into the Univac I. Each six-bit character was augmented by a seventh parity bit. Words moving between memory and registers were checked by a parity detector associated with the processor-memory bus (high speed bus amplifier). Every five seconds, processing would be stopped to circulate the contents of memory through the parity checker to ensure integrity of information stored in the memory. This is perhaps the first use of "memory scrubbing" recorded. Integrity of control flow was maintained by carrying parity through the instruction buffer and into the instruction register. The gate control signals produced by the function table were considered to be one large "word" to which parity was added and checked. Extra dummy signals were added to the function table to ensure proper parity. Parity was also appended to the characters retrieved from secondary storage (e.g., tape).

While parity is an effective code for detecting transmission errors, it is more difficult to use when the data

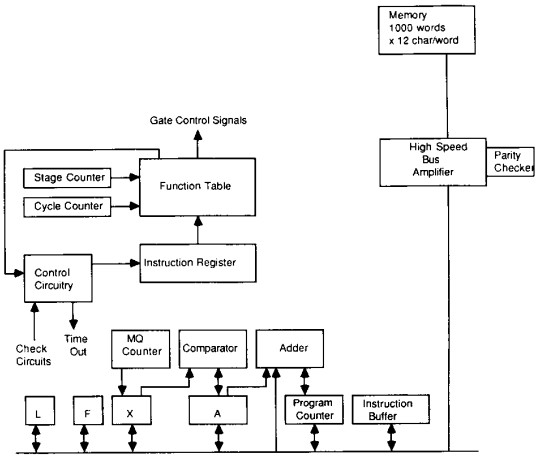

Fig. 10. Block Diagram of Univac 1.

Table 6 Error Detection Features in Univac 1

Parity	
	Instruction Buffer
	Instruction Register
	Function Table Output
	Memory
	Tape
Duplication	
	Adder
	Comparator
	A, F, X, L Registers
	Cycle Counter
	Stage Counter
	MQ Counter
	High Speed Bus Amplifier
1-of-N	
	Intermediate Function Table Groups
	Memory bank select
Block Length Counter	
	Tape

is transformed such as by arithmetic operations. Thus the Univac I designers made extensive use of duplication in the arithmetic units as well as the counters controlling the sequencing of pulses. The major arithmetic registers were also duplicated. A disagreement between the two copies of any resource would halt the machine.

When control information is decoded, typically only one or no signals in a group are turned on. One-of-n checkers were used on intermediate fields of the function table as well as the memory bank selection circuitry. Finally the number of characters in a block transmitted from secondary storage were carefully counted to detect incomplete or overrun transmissions.

As we can see, the Univac I contained more error-detection circuitry than the majority of contemporary minicomputers and microprocessors. These error detection circuits were essential in order to debug the complex design since there were no simulators to verify concepts. The error detection circuits also provided confidence that the computational results produced by the Univac I were correct.

2) IBM: Table 7 presents the evolution of IBM's maintenance strategy. Techniques are listed for a representative machine from each major era. The techniques can be loosely grouped in three major categories: internal hardware error-detection circuits, diagnostics (including software and microcode), and display (such as lights, error logs, tracing). The IBM strategy has evolved from "failure recreation" to "failure capture," to "failure recover." Prior to the S/370, IBM customer engineers attempted to recreate the failure by running diagnostics, sometimes in conjunction with varying voltage and clock frequency, until the failure reoccurred. The system was placed in a tight programmed loop to produce a continuous failure condition for analysis. In failure capture, hardware circuits detect errors and information about the current status of the machine state is logged for subsequent analysis. In failure recover, the information capture by the detection circuits is utilized by recovery techniques, in hardware, microcode, and software, to allow the application or job to complete. The information is also logged at a remote support facility to assist engineering design activity. Historical perspectives of IBM's maintenance strategies can be found in [43], [44].

Table 8 lists the feature in the IBM 3090 series. The hardware error-detection, error-correction, and monitoring circuits described in Table 8 are used in the following maintenance scenario. The processor controller analyzes the error detection data and develops a field replacement unit (FRU) call or a maintenance action plan. Automatically, at the customer's option, the processor controller establishes a data link for service. A customer engineer receives the FRU information and responds to the site with the appropriate part or parts needed to effect the repair. For additional information the customer engineer can communicate via data link with a central data base (called RETAIN) for the latest service tips. A Field Technical Support Center specialist can use the data link to monitor remotely and/or control diagnostics on the IBM 3090. The reliability, availability, and serviceability (RAS) features of the IBM 308X and 3090 Processor Complexes can be found in [1], [45], [46]. The goal of both hardware and software is high availability with minimized impact of failures. Four stages of corrective action are identified, each with successively larger impact on users: transparent recover, one user affected, multiple users affected, and down. The successively higher-severity stage recovery structure is common in systems with high-availability goals or in real-time data processing environments in which temporary loss of data is tolerable. The IBM strategy of "first failure capture" is evident in the design of both hardware and software.

Error detection and recovery in general-purpose computers have evolved a long way from the Univac I design. The evolution is continuing into techniques for multiple computer and microprocessor organizations. However we will continue this discussion with high availability systems.

Table 7 Evolution of IBM Maintenance Strategy

Machine	Era	Techniques
650	Late 1950's	Six internal checkers
		Stand-alone diagnostics on punched cards
		Light and switch panel
1401	Early 1960's	20 internal checkers
		Stand-alone diagnostics
		Light and switch panel
S/360-50	Mid-1960's	75 internal checkers
		OLTEP—On-line Test Executie Program
		Microdiagnostics
		Log fault data to main memory
		EREP—Error Recording and Edit Program for outputing logged data
		Maintenance panel
370/168 Mod 3	Early 1970's	Error-detection circuits
		OLTEP
		Microdiagnostics for fault isolation
		Service processor, including trace unit—trace up to 199 fixed and 8 movable logic points over 32 machine cycles for intermittent or environmental faults
303X	Mid-1970's	Error-detection circuits
		OLTS—On-Line Functional Tests
		Console and processor microdiagnostics
		EREP
		Scope loops
		Support processor, including trace and remote (telephone) access to log data and trace information
4341	Late 1970's	Error-detection circuits
		25 000 shadow latches
		Support processor—error logging and environmental monitoring
3090	Mid-1980's	Error detection circuits
		Error correction circuits
		Recovery techniques
		Console and processor microdiagnostics
		Processor controller, duplicated for availability
		Fault isolation circuits
		Fault threshold and isolation analysis
		Autocall and remote access to service information and remote service console
		EREP
		On-line tests
		Reconfiguration capability
		Internal machine environment monitoring

B. High Availability

1) AT&T: The AT&T Switching Systems pioneered fault-tolerant computing in the telephone switching application. Table 9 traces the variations of duplication and matching devised for the switching systems to detect failures and to automatically resume computations. The primary form of detection is hardware lock-step duplication and comparison requiring about 2.5 times the hardware cost of a nonredundant system. Even though all the processors are based upon full duplication, it is interesting to observe the evolution from tightly lock-stepped matching every machine cycle in the early processors to a higher dependence on self-checking and matching only on writes to memory. Furthermore, as the processors evolved from dedicated,

Table 8 IBM 3090 Series RAS (Reliability, Availability, Serviceability)

Reliability

- Low intrinsic failure rate technology
- Extensive component burn-in during manufacture
- Dual processor controller that incorporates switchover
- Dual Direct Access Storage units support program controller switchover
- Multiple consoles for monitoring processor activity and for backup
- LSI Packaging reduces number of circuit connections
- Internal machine power and temperature monitoring
- Chip sparing in memory replaces defective chips automatically

Availability

- Two, four, or six central processors
- Automatic error detection and correction in central and expanded stroage
  - Single bit error correction and double bit error detection in central storage
  - Double bit error correction and triple bit error detection in expanded stroage
- Storage deallocation in 4K-byte increments under system program control
- Ability to vary channels off-line in one channel increments
- Instruction retry
- Channel command retry
- Error detection and fault isolation circuits provide improved recovery and serviceability
- Multipath I/O controllers and units

Data integrity

- Key controlled storage protection (store and fetch)
- Critical address storage protection
- Storage error checking and correction
- Processor cache error handling
- Parity and other internal error checking
- Segment protection (S/370 mode)
- Page protection (S/370 mode)
- Clear, reset of registers and main storage
- Automatic Remote Support authorization
- Block multiplexer channel command retry
- Extensive I/O recovery by hardware and control programs

Serviceability

- Automatic fault isolation (analysis routines) concurrent with operation
- Automatic remote support capability–auto call to IBM if authorized by customer
- Automatic customer engineer and parts dispatching
- Trace facilities
- Error logout recording
- Microcode update distribution via remote support facilities
- Remote service console capability
- Automatic validation tests after repair
- Customer problem analysis facilities

real-time controllers to multiple purpose processors, the operating system and software not only became more sophisticated but also became a dominant portion of the system design and maintenance effort [47].

The AT&T Switching Systems had an aggressive availability goal of 2 hours downtime in 40 years (3 minutes per year). The 3B20D processor, used in the No. 5 ESS, has been released as a commercial product. As shown in Fig. 11, every function is duplicated. The duplicated CPU's are self-checking, each with error-correcting memory. One CPU is active, the other is standby. The contents of both memories are kept identical through use of a memory update link. Disks are duplicated and dual-ported. Disks can also be mirrored (two identical copies of files on two independent disk drives). When a failure occurs, the operating system consults a database to determine an alternate configuration of hardware units. If a fault is detected in the active unit, the standby is switched in. Further

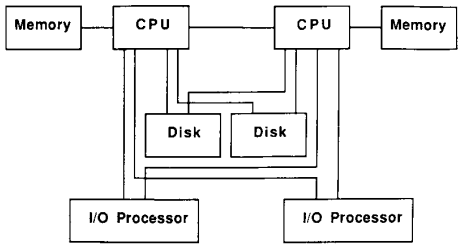

Fig. 11. Architecture of 3B20D duplex system.

information on AT&T electronic switches can be found in [11], [48]–[67].

2) Tandem: Over a decade after the first AT&T computer-controlled switching system was installed, Tandem designed a high-availability system targeted for the on-line transaction processing (OLTP) market.

Table 9 Summary of ESS Processors

Processor	Year Introduced	Complexity (gates)	Unit of switching	Matching	Other error detection/correction
No. 1	1965	12 000	Program Store (PS) Control Store (CS) Central Controller (CC)	Six internal nodes, 24 bits per node; one node matched each machine cycle, node selected to be matched dependent on instruction being executed	Hamming codes on PS Parity on CS Automatic retry on CS, PS Watchdog timer Sanity program to determine if reorganization led to a valid configuration
No. 2	1969	5 000	Entire computer	Single match point on call store input	Diagnostic programs Parity on PS Detection of multiword accesses in CS Watchdog timer
No 1A	1976	50 000	PS, CS, CC, buses	16 internal nodes, 24 bits per node, four nodes matched each machine cycle	Two parity bits on PS Roving spares (i.e., contents of PS not completely duplicated, can be loaded from disk upon error detection) Two parity bits on CS Roving spares sufficient for complete duplication of transient data Processor configuration circuit to search automatically for a valid configuration
No. 3A	1975	16 500	Entire computer	None	On-line processor write into both stores m-of-2m code on microstore plus parity Self-checking decoders Two parity bits on registers Duplication of ALU Watchdog timer Maintenance channel for observability and controllability of the other processor 25% of logic devoted to self-checking logic and 14% to maintenance access.
3B20D	1981	75 000	Entire computer	None	On-line processor write into both stores Byte parity on data paths Parity checking where parity preserved, duplication otherwise Modified Hamming code on main memory Maintenance channel for observability and controllability of the other processor 30% of control logic devoted to self-checking Error correction codes on disks Software audits, sanity timer, integrity monitor

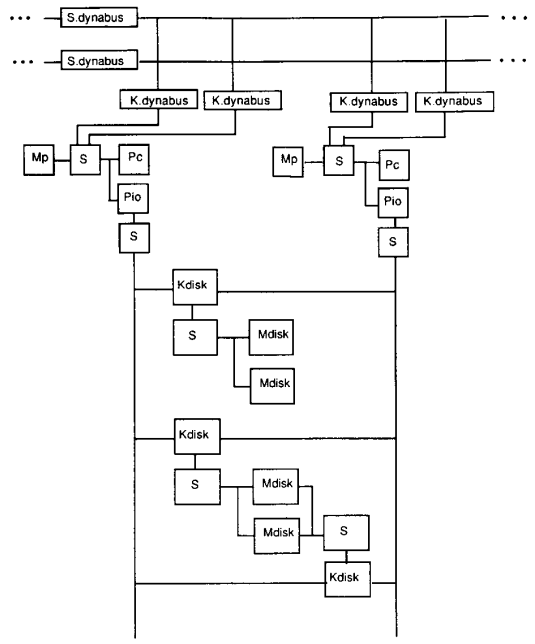

Fig. 12. Tandem architecture.

Several architectural features such as: dual paths to all system elements (including disks and I/O controllers), processor replication, redundant power supplies, mirrored disks, and a message-based operating system were used to not only tolerate failure, but also to provide modular expansion of computer resources.

The Tandem architecture, as shown in Fig. 12, has remained relatively constant since it's inception. The Tandem system is composed of from 2 to 16 computers, each with its own memory and I/O channels, interconnected by a message-oriented 16-b, high-performance, dual interprocessor Dynabus. Since processors are individually powered, a processor can be shut down without affecting either its corresponding processor or other system components. Each processor maintains its own copy of the operating system.

The architecture has evolved through five different processors, representing a factor of four improvement in performance, and two different Dynabuses, yielding a factor of 1.5 improvement in throughput.

Tandem was concerned about propagation of errors and thus developed a loosely-coupled multicomputer architecture. While one computer acts as primary, the backup computer is active only to receive periodic checkpoint information. Hence 1.3 physical computers are required to behave as one logical fault-tolerant computer. Tandem relies upon both hardware (e.g., data path parity, error correcting code memory, watchdog timers, and Dynabus message checksums) and software to detect and recover from failure. The primary data integrity mechanism is "process pairs."

Disk data is accessed by virtue of user processes interacting with disk I/O processes [68]. Data transfer is effected via the device process that is responsible for controlling the particular physical device on which a file resides. Disk files are duplicated on physically distinct devices controlled by an I/O process pair. Thus in the event of physical failure or isolation of the primary, the backup file is up to date and available.

A process pair consists of two copies of a process, each residing in a separate physical processor. Processors can be teamed together, through software, in arbitrary pairs. One process is designated as primary, and the other is designated as a semiactive backup. If there is a failure of the primary's processor or of the access paths to the I/O devices, the backup can take over. The primary process sends an "I'm alive" signal to the backup process once a second. If the backup process fails to receive the "I'm alive" signal within two seconds an exception is generated and the backup takes over as the primary. When the primary resumes on-line status, the unexpected "I'm alive" message at the backup (temporarily primary) triggers another exception, and the role of primary and backup are again swapped. Since mirrored disk volumes may get out of synchronization during the period in which the backup assumes the primary role, provisions are made for gradual resynchronization of the mirrored disk volumes as the system recovers from a processor failure.

Checkpoint information is sent, via message passing, from the primary to the backup. The checkpoint information is inserted in the corresponding memory locations of the backup process, as opposed to the more usual approach of updating a disk file. This approach permits the backup process to take over immediately in the event of failure without having to perform the usual recovery journaling and disk accesses before processing resumes. It is interesting to note that the Tandem approach relies heavily on software to bring fault tolerance to hardware that, by itself, would not be fault tolerant.

More details on the Tandem operating system and the programming of nonstop applications can be found in [69]–[72]. Other transactions-oriented processors are described in [39], [73], and [74].

3) Stratus: Stratus Computers, Inc., founded in 1980, uses an alternative to the Tandem approach for on-line transaction processing by employing single chip microprocessors. The design goal is for continuous processing which Stratus defines as uninterrupted operation without loss of data, performance degradation, or special programming. Stratus utilizes continuous checking between duplexed components for detection of errors at the actual point of failure.

The Stratus self-checking, duplicate-and-match architecture is shown in Fig. 13. A module (or computer) is composed of replicated power and backplane buses (StrataBus) into which a variety of boards can be inserted. Boards are logically divided into halves which drive outputs to and receive inputs from both buses. The bus drivers/receivers are duplicated and controlled independently. The logical halves are driven in lock-step by the same clock. A comparator is used to detect any disagreements between the

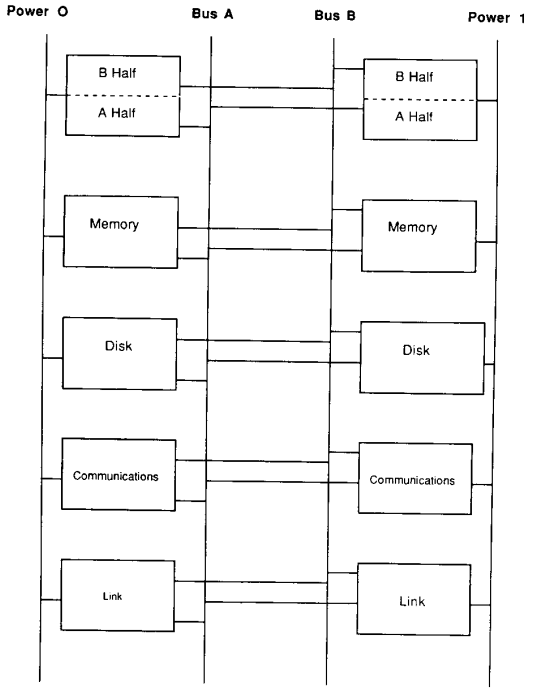

Fig. 13. The Stratus pair-and-spare architecture.

two halves of the board. Multiple failures which affect the two independent halves of a board could cause the module to hang as it alternates between buses seeking a fault-free path. Up to 32 modules can be interconnected into a system via a message passing Stratus intermodule bus (SIB). Access to the SIB is via dual 14 megabyte per second Links. Systems, in turn, are tied together via an X.25 packet switched network.

Now consider how the system in Fig. 13 tolerates failure. The two processor boards (each containing a pair of microprocessors) are each self-checking modules used in a "pair and spare" configuration. Each board operates independently. Each half of each board (e.g., Side A) receives inputs from a different bus (e.g., Bus A) and drives a different bus (e.g., Bus A). Each bus is the wired-OR of one half of each board (e.g., Bus A is the wired-OR of all "A" board halves). The boards constantly compare their two halves and upon disagreement, the board removes itself from service, a maintenance interrupt is generated, and a red light is illuminated. The spare pair on the other processor board continues processing and is now the sole driver of both buses. The operating system executes a diagnostic on the failed board to determine whether the error was due to a transient or permanent fault. In the case of a transient, the board is returned to service. Permanent faults are reported by phone to a customer assistance center (CAC). The CAC reconfirms the problem, selects a replacement board of the same revision, prints installation instructions, and ships the board by overnight courier. The first time the user realizes there is a problem is when the board is delivered. The user removes the old board and inserts the new board without disrupting the system (i.e., "hot" swap). the new board interrupts the system and the processor which has been running brings the replacement into full synchronization at which point the full configuration is available again. Detection and recovery are transparent to the application software. Similarly to Tandem, the Stratus architecture has remained essentially constant evolving through four different microprocessors. More details on the Stratus architecture can be found in [75].

4) Intel 432: The Intel 432 was designed as a set of building blocks from which high-availability and high-reliability systems could be built. By adding a small amount of extra hardware to a VLSI chip set, a wide range of system types are possible. The Intel 432 system pioneered many concepts that appear in contemporary microprocessors.

The basic components in the Intel 432 are shown in Fig. 14 along with the fault-confinement regions. In order to limit the spread of errors, carefully defined fault-confinement regions were designed, primarily at chip boundaries. Hardware checks data as it leaves a confinement region. If an error is detected it assumes that the fault is contained in this fault-confinement region.

Figure 15 depicts a typical Intel 432 system using duplication and matching as the primary fault-detection mechanism. One processor is designated master by enabling a pin on the processor chip. Only the master processor drives the bus, but both processors receive data from the bus. Thus the checker receives data from the bus, performs the appropriate internal computation, and compares its output to that which the master has put on the bus. This functional redundancy checking is also used on the memory array controllers. The memory arrays are not duplicated, rather they depend on error-correcting codes to tolerate failures.

Automatic recovery from processor and/or memory failure uses the primary/backup matching scheme employed by Stratus. Error detection, reporting, and recovery mechanisms are periodically exercised through software that forces error conditions and waits to observe the results. More information on the Intel 432 architecture can be found in [76] and [77]. Note that the Intel 432 building blocks could be used to fabricate systems in all three branches of the fault-tolerant techniques space.

C. Critical Computations

1) C.vmp: Masking redundancy is the primary technique used for critical computations and long life applications. Because by definition, masking redundancy corrects faults before they reach module outputs, detection and recovery are part of the same process. Pure fault masking thus gives no warning of a deteriorating hardware state until enough faults have accumulated to cause a failure. Because of this problem most fault-masking techniques are extended to provide fault detection as well. The additional redundancy needed for this purpose is usually only a minor increase.

On uniprocessors, fault masking can take the form of either temporal or spatial replication. In principle, any uniprocessor can be programmed to perform a computation

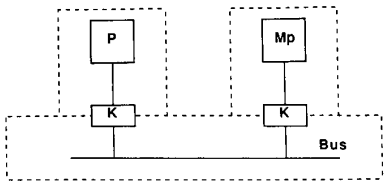

Fig. 14. Basic building block of the Intel 432 with fault confinement regions indicated.

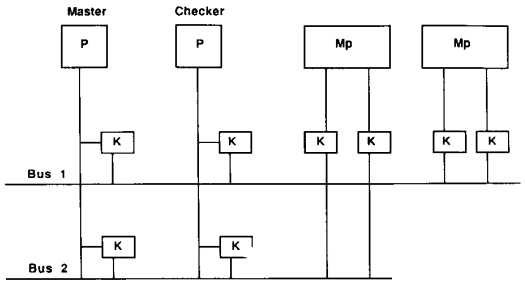

Fig. 15. Functional redundancy checking in the Intel 432.

three times, followed by a vote to calculate the majority results. Of course, software voting on a uniprocessor will take over three times as long as the nonredundant computation.

The hardware approach to fault masking in uniprocessors can be considered as a logical extension to duplication and comparison. Rather than compare the outputs of two copies at each clock tick, the majority results of three copies are produced by a voting circuit.

Figure 16 depicts the architecture of C.vmp, which uses off-the-shelf components and a custom-designed voter to achieve fault-tolerance. Voting occurs every time the processors access the bus to either send or retrieve information. C.vmp in fact consists of three separate machines capable of operating in independent mode and executing three separate programs. Under the control of an external event or under the control of one of the processors, C.vmp can synchronize its redundant hardware and start executing the critical section of code. With the voter active, the three buses are voted upon and the result of the vote is sent out. Any disagreements among the processors will therefore not propagate to the memories and vice versa. Because voting is a simple act of comparison, the voter is memoryless. Voting is done in parallel on a bit-by-bit basis. A computer can have a failure on a certain bit in one bus, and, provided that the other two buses have the correct information for that bit, operation will continue. There are cases, therefore, where failures in all three buses can occur simultaneously and the computer would still be functioning correctly.

Bus-level voting works only if information passes through the voter. Any internal state of the processor must periodically circulate through the voter to clean up errors (i.e., scrubbing as pioneered in Univac I). Traces of actual programs indicate that the internal processor state is frequently written to memory during normal program and operating system behavior. C.vmp runs an unmodified version of

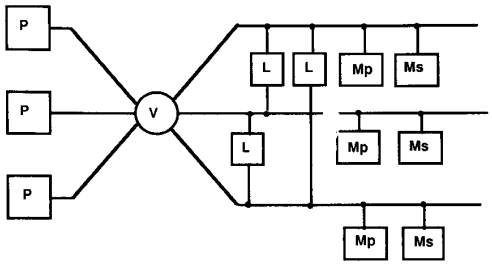

Fig. 16. Architecture of C.vmp.

a commercial operating system. Thus the fault-tolerant properties of the architecture are achieved in a manner transparent to the software (i.e., no modifications need to be made in the operating system or application software). In addition to the obvious overhead of having three copies of all the components, actual measurements on the system indicate a further degradation of 15% in performance due to the added delays induced by the voter sitting between the processor and memories. More information on C.vmp can be found in [78]. The Saturn V launch vehicle computer used triplication and voting [79]. The Fault-Tolerant Multiprocessor (FTMP) allow processor and memory triads to be dynamically formed from a pool of resources [80].

2) SIFT: The performance of software voting is improved if there is a separate processor dedicated to each copy of the software task. Furthermore the potential bottleneck of the hardware voter can be removed by only periodically performing the voting function in software.

SRI International designed Software Implemented Fault-Tolerance (SIFT) [81]–[83] for real-time control of aircraft. The hardware consists of independent computers communicating with other computers over unidirectional serial links. Thus for N computers there are $N(N-1)$ links. The SIFT software is divided into a set of tasks. The input to a task is produced by the output of a collection of tasks. Reliability is achieved by having tasks done independently by a number of computers. Typically, the correct output is chosen by a majority vote. If all copies of the output are not identical, an error has been detected. Such errors are recorded for use by the executive for determining faulty units and system reconfiguration. Voting is performed only on the input data to tasks rather than on every partial result. Thus the tasks need to be only loosely synchronized (e.g., to within 50 microseconds).

Figure 17 depicts the distribution of tasks among three processors. Application task A receives its input from task B. Task A receives the majority voted input from three copies of task B provided by the executive. When task A finishes, it places its output into a buffer so that the executive can provide the majority voted data as input to the next task. The number of processors executing a task can vary with the task and even with the execution instance of the task. As depicted in Fig. 17, the tasks reside on different computers. A commercial version of SIFT was offered by August Systems [84].

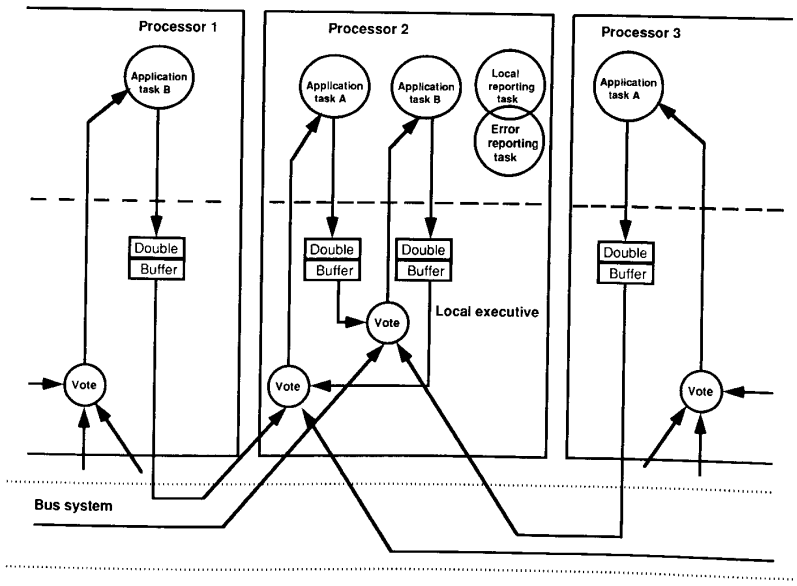

Fig. 17. Arrangement of application tasks within SIFT configuration [Adapted from 83].

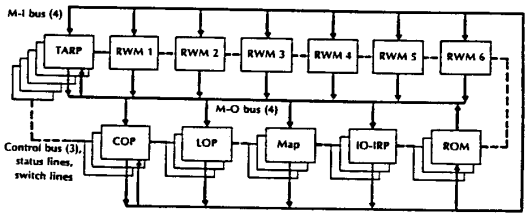

Fig. 18. STAR computer organization [Adapted from 38].

D. Long Life

1) STAR: Spacecraft are the primary example of systems requiring long periods of unattended operation. Unlike most other applications, spacecraft must directly control their environment, including electrical power, temperature, and even stability. Thus one must treat all aspects of a spacecraft (e.g., structural, propulsion, power, analog, and digital) when designing for reliability.

Spacecraft missions cover a range from simple (e.g., weather satellites in low earth orbit) to sophisticated (e.g., deep-space planetary probes passing through uncharted environments). Some points in the range are low-earth-orbit sensing, low-earth-orbit communications or navigation, low-earth-orbit scientific, synchronous-orbit communications, and deep-space scientific. For nondemanding missions, system reliability goals are met through reducing complexity and simplicity of design.

A typical maintenance procedure would be as follows. When a failure has been detected, the spacecraft automatically enters a "safe" or "hold" mode. All nonessential loads on the power subsystem are shed. Normal mission sequencing and solar array tracking are stopped. The spacecraft is oriented to obtain maximum solar power. Meanwhile, the ground personnel must infer what possible failures could cause the output behavior of each of the component strings. A possible failure scenario is selected as most likely and a reconfiguration (e.g., "work around") of the spacecraft subsystems devised. A command sequence implementing the "work around" is sent to the satellite. Depending on the severity of the failure, this entire procedure may take days, or even weeks, to complete.

Spacecraft fault responses vary from automatic in hardware for critical faults (e.g., power, clocks, and computer), to on-board software for serious faults (e.g., attitude and command subsystems), to ground intervention for noncritical faults.

One of the more demanding deep space missions was the "Grand Tour" wherein a single spacecraft would use a rare alignment of the planets in the solar system to make a fly-by of the five outer planets. A mission life of ten years was required. In support of the Grand Tour mission, the Jet Propulsion Laboratory (JPL) designed and breadboarded a self-test and repair (STAR) computer [38].

Figure 18 depicts the architecture of the STAR computer. Data communication between the units is over two four-wire buses: memory-out (M-O) and memory-in (M-I) bus. There are both read-only (ROM) and read/write (RWM) memories; separate processors for instruction sequencing (COP), logical operations (LOP), arithmetic operations (MAP), I/O (IOP), and interrupts (IRP); and a test and repair processor (TARP). The philosophy used in STAR was to detect errors at interfaces to blocks and replace blocks upon failure. Redundancy techniques were used that were appropriate to each block. Data are composed of 28-b words with an appended 4-b residue code, transmitted in

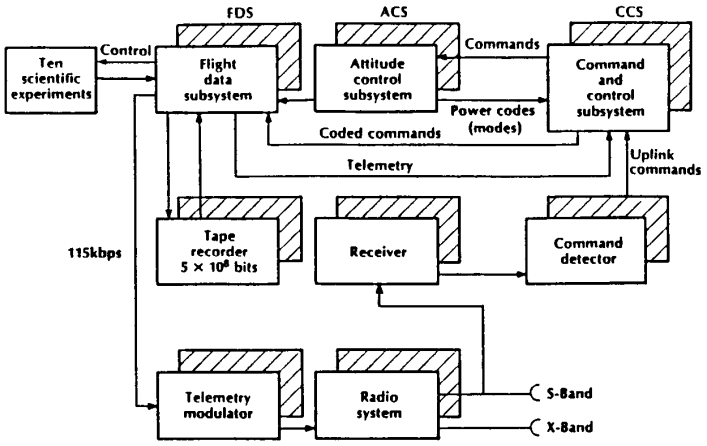

Fig. 19. Voyager system block diagram.

eight successive nibbles over the four-wire buses. Instruction operation codes are divided into three bytes, each encoded as a 2-out-of-4 code. Duplication is used to detect errors in the logic processor and read/write memories. Each unit reports two bits of status (e.g., unit output active, disagree with bus for duplicate units, operation completed, and internally detected fault) to the TARP. The TARP monitors both the memory buses for correctly encoded information and the status bits from the various units. The unit status bits can only assume specific sequences during fault-free operation. Any deviation indicates an error. Upon error detection the TARP directs reconfiguration—and recovery. Reconfiguration is achieved by power switching of spare units. Recovery is accomplished by resetting the program counter and internal registers to values stored by software at user-specified "rollback points." The TARP is critical to the error detection, reconfiguration, and recovery activities and hence is implemented in hybrid redundancy [85] with three active, majority-voting copies and two standby spares that can replace any copy that disagrees with the majority output. Due to budgetary constraints, STAR was never flown. Two Voyager missions were modified to achieve most of the objects of the Grand Tour mission.

2) Voyager: Figure 19 displays the interconnection of subsystems on the Voyager spacecraft. Standby redundancy is used in all but the sensor payload [86]. The standby spares are "cross-strapped" so that either unit can be switched in to communicate with other units. This form of standby redundancy is called "block" redundancy in that redundancy is provided at the subsystem level rather than internal to each subsystem. The attitude control subsystem (ACS) is composed of redundant computers; one is an unpowered standby spare. The command and control subsystem (CCS) is also a redundant computer, but the standby is powered and monitors the on-line unit. Cross-strapping and switching allow reconfiguration around failed components. The CCS executes self-testing routines prior to issuing commands to other subsystems. Tables 10 and 11 list the error detection mechanisms in the Voyager attitude control and command control subsystems. Memory is only 4K words. The tape recorders are used for storage of scientific data only. New programs for memory have to be loaded from the ground. The Voyager spacecraft returned close-up photographs of Saturn, Jupiter, Uranus, and Neptune over a 12-year mission spanning over four billion miles. Several spacecraft failures were tolerated in successfully completing their mission.

Table 10 Error Detection in Voyager Attitude Control Subsystem

CCS fails to receive "I'm Healthy" report every 2 seconds
Loss of celestial (sun and Canopus) reference
Power supply failure
Fail to rewrite memory every 10 hours
Spacecraft takes longer to turn than expected (thruster failure)
Gyro failure
Parity error on commands from CCS
Command sequence incorrect
Failure to respond to command from CCS

Table 11 Error Detection in Voyager Command and Control Subsystem

Hardware	
	Low voltage
	Primary command received before preceding one processed
	Attempt to write into protected memory without override
	Processor sequencer reached an illegal state
Software	
	Primary output unit unavailable for more than 14 seconds
	Self-test routine not successfully completed
	Output buffer overflow

3) Galileo: A follow-on to the Voyager Jupiter fly-by mission is the Galileo Jupiter orbiting and probe insertion mission. Figure 20 depicts a block diagram of the major subsystems in the Galileo orbiter. As can be seen, the Galileo architecture borrows heavily from the experiences

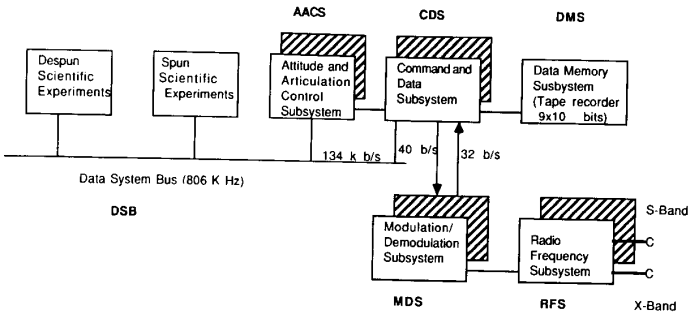

Fig. 20. Galileo orbiter block diagram.

gained with the Voyager system. Block redundancy is used throughout the subsystems comprising the orbiter. All but the command and data subsystem (CDS) operate as an active/standby pair. The CDS operates as active redundancy wherein each block can issue independent commands or they can operate in parallel on the same critical activity. The major departure from the Voyager architecture is the extensive use of microprocessors in the Galileo orbiter. A total of 19 microprocessors with 320 Kbytes of random access memory and 41 Kbytes of ROM form a distributed system communicating over an 806 kilohertz data bus. Scientific instruments add eight further microprocessors to the total system. The bus is used to pass network-like messages between sources and destinations. Due to the volatile nature of the random access memory, a further requirement for the Galileo orbiter was memory keep-alive inverters to maintain power to the command and data subsystem (CDS) and attitude and articulation control subsystem (AACS) memories in case of power faults. The orbiter accommodates a total of nine scientific instruments (five fields and particles and four remote sensing science). Due to the nature of the phenomena to be measured, fields and particles instruments demand a spinning platform to make total spherical observations while the remote science instruments require very accurate and very stable pointing. These requirements produced a dual-spin structure wherein a portion of the spacecraft is spun at three to ten revolutions per minute while the other portion is held in a stable fixed configuration. Power is transmitted across the spun/despun interface via slip rings while rotary transformers are used for data signals.

The down link data rates vary as a function of the experiment. Nonimaging science experiments require a high quality bit-error rate (less than $5 * 10^{-5}$) and are encoded in a Golay (24, 12) error-correcting code. Imaging experiments can use a lower quality bit-error rate (less than $5 * 10^{-3}$). The orbiter has been designed to operate reliably and autonomously in the harsh Jovian radiation and electrostatic discharge environments during the critical phase of relaying probe data and orbit insertion.

The Galileo spacecraft has few hardware error detection mechanisms. Faults are detected by monitoring the performance of various spacecraft subsystems. A partial list of error detection mechanisms include [87] the following:

- test of event durations including transfers between subsystems, transition between all spin and spun/despun modes;
- parity or check sum errors on messages;
- unexpected command codes;
- loss of "Heartbeat" between the AACS and the CDS;
- spin rates above or below set values;
- loss of sun or star identification detected by no valid pulse from acquisition sensor for a given period of time;
- error between control variable setting and measured response is too great.

The on-board fault protection software is designed to alleviate the effects and symptoms of faults rather than to pinpoint the exact faults themselves. Fault identification and isolation is performed by ground intervention.

The Galileo mission was scheduled to fly from the space shuttle in May of 1986. However, the Challenger explosion postponed the launch until 1989.

VII. Historical Perspective of Fault-Tolerant Architectures

Before attempting to extrapolate trends in fault-tolerant architectures, we must increase the population size of our design space. Table 12 presents highlights of several other architectures which will be used in our study. A brief description of each architecture follows:

- Commercial: Commercial general-purpose computing families including the DEC VAX [88], and UniSys [89]–[91] employ many of the same techniques described previously for the IBM family of computers.
- Sage: Sage was developed by IBM for the air defense system. Composed of 55 000 vacuum tubes, the Sage processor implemented elaborate fault-detection schemes consisting of parity and software tests [92]. The stand-by computer executed frequent self-tests and its memory was periodically updated by the on-line computer. After error detection, switchover and recovery were executed by software. Sage served for 25 years until the last installation was retired in 1983.
- RW-40: Ramo-Wooldridge dual processor system.

Table 12 Feature of Other Fault-Tolerant Systems

General Purpose	
Commercial	Parity on Date Paths
General Purpose	Parity on Control Stores
Systems	Error Correcting Code on Memory
	Instruction Retry
High Availability	
Sage	Standby Processor
RW-40	Standby Processor
Pluribus	Periodic Software Checks
	Watchdog Timers
	Redundant Data Structures
	Parity
	Checksum
Parallel	Dynamic Duplication
Sequoia	Parity
	Error Correcting Code
	Duplication
	Watchdog Timer
Critical Computations	
Saturn V	Triplication and Voting
FTP	Four copies in lockstep synchronization
	Hardware voting
Space Shuttle	Software voting
	Five computers
	Design Diversity
AIPS	Distributed System
	Voting on system outputs
Long Life	
OAO	Quadded components

- Pluribus: A modular multiprocessor system composed of processor buses, memory buses, and I/O buses. It served as a message-switching processor in the ARPANet. Remote diagnosis is performed by the Network Control Center. Even in the case of total destruction of the contents of memory, a Pluribus can request the code be transmitted from the Network Control Center or other Pluribuses in the system. Any transitory messages lost are restored via the retransmission mechanism in the network protocol. More information can be found in [93]–[97].
- Parallel: A duplicated architecture construction from industrial standards such as the Multibus, Motorola 68000 processors, and the Unix operating system. All tasks are executed simultaneously on both processors. Upon fault detection the faulty processor is removed by the operating system and the application continues to run on the second processor.
- Sequoia: A modularly expandable multiprocessor employing error correcting codes, duplication, and protocol monitoring [98]. Caches, buffers, and buses are protected by parity. The main memory employs error correcting code. Every processor, memory, and I/O processor is duplicated with several internal points of comparison every clock cycle. If a mismatch is detected, the element electrically isolates itself from the rest of the system. Code and unmodified data are backed up on disks so that the contents of a failed memory module can be reconstructed. Disks are also mirrored to tolerate failures.
- Saturn V: The launch vehicle computer for the Saturn V rocket [79]. Triplication and voting is used for functional modules while a back-up sparing technique is employed for memory. The memory operates in duplex mode. Memory errors are detected by parity and/or monitoring of memory access line drive current. If an error is detected in the on-line memory, operation is transferred to the standby memory without interruption of service or loss of data.
- FTP: A Fault-Tolerant Processor (FTP) utilizing four copies in lock-step synchronization to tolerate failures [99].
- Space Shuttle: The space shuttle computer [100]–[102] uses four of its five computers as a primary system. The four primary computers perform identical computation and broadcast their results. The outputs of the four primary computers are voted on at the control actuators. In addition, each computer listens to the outputs of the three other computers and compares their signals with its own via special software. If a computer detects a disagreement, it signals the disagreeing computer. The received disagreement detection signals are voted on in the redundancy management circuitry of each computer. If the vote is positive, the redundancy management unit removes its computer from service. Up to two computer failures can be tolerated in voting mode operation. After the second failure, the system converts to a duplex system that can survive one additional computer failure by using comparison and self-test methods. When the third failure occurs, it is detected but is uncertain as to which computer has failed. The space shuttle is said to have a "fail safe, fail safe, fail soft" system. The fifth computer contains a back-up flight software package written by a different contractor. The fifth computer represents a "sanity check" on the other four and guards against common mode failure (i.e., program bug) in the software of the four primary computers. The use of two dissimilar designs is called "design diversity."
- AIPS: A reconfigurable, distributed system in which the individual nodes can be uniprocessors or fault-tolerant processors such as FTP [103].
- OAO: Orbiting Astronomical Observatory (OAO) which uses four copies of discrete components such as resistors, capacitors and transistors arranged in a network which can tolerate any single open or short circuit.

The three-dimensional space of application, architecture, and technique presented in Table 5 has been broken into the application/architecture and application/technique subspaces in Figs. 21 and 22, respectively. The system

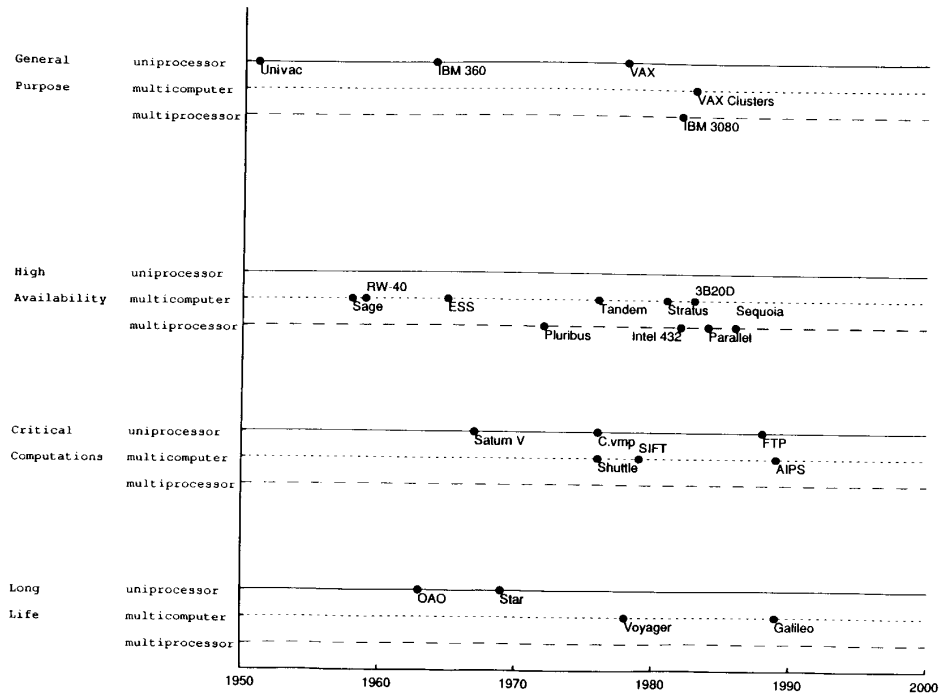

Fig. 21. Historical perspective of the application/architecture dimensions.

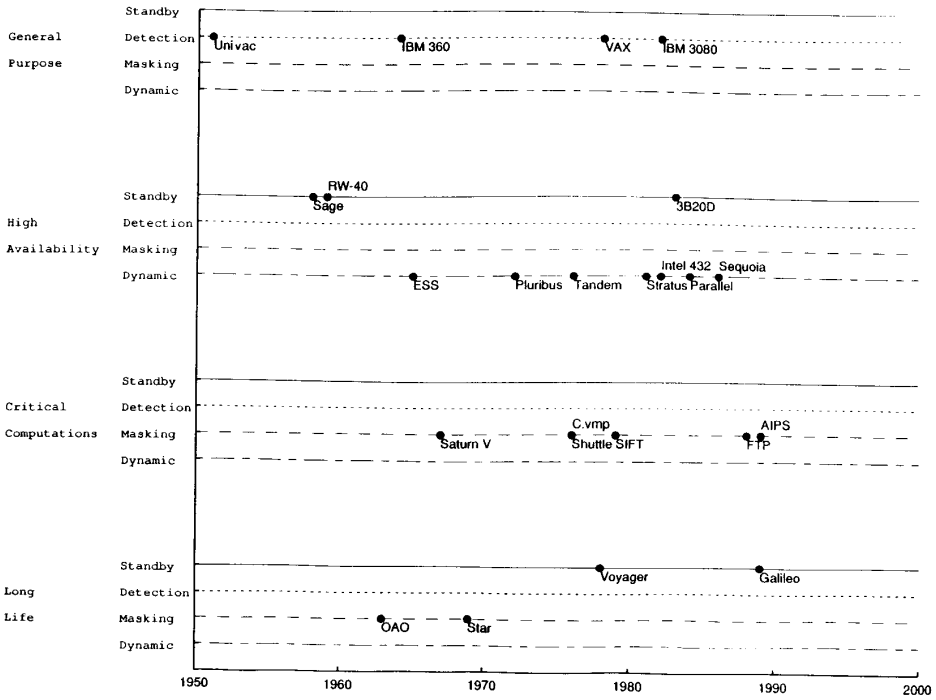

Fig. 22. Historical perspective of the application/techniques dimensions.

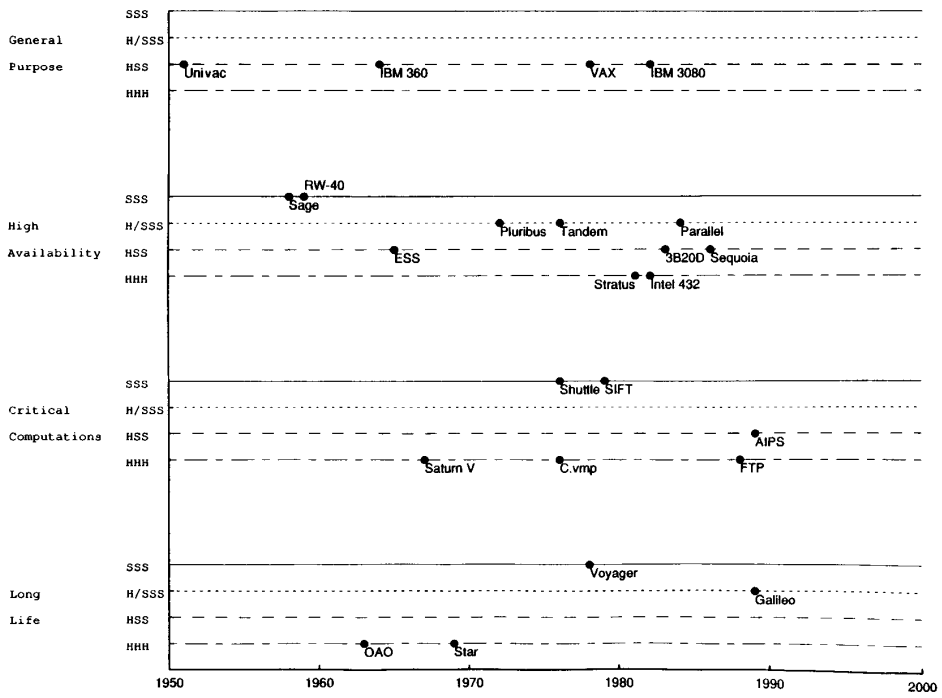

Fig. 23. Historical perspective of the application/fault handling approaches.

examples given in Table 5 are plotted in Figs. 21 and 22 by the year of introduction. Figure 23 plots the same systems by application type and whether they use hardware/software for three of the major stages in fault handling (detection, isolation, and recovery). With these figures taken as a group, several interesting trends can be noted:

- General-purpose processors have focused on hardware detection with software isolation and recovery. These architectures have also been moving from uniprocessors to multicomputers and multiprocessors. Error detection is in hardware with isolation and recovery performed by software and microcode.
- High-availability systems have started with multicomputers with a trend toward multiprocessors. Techniques started with stand-by while dynamic redundancy is almost universal today. The trend is also to perform more of the stages of fault handling in hardware commencing with stand-by redundancy effectively employing all software techniques to completely hardware-based systems such as Stratus and Intel 432.
- Critical computations have all employed masking techniques with all hardware approaches for uniprocessors to tolerate hardware failures and predominantly software approaches in multicomputers to tolerate design errors. There is a blending of the hardware and software approaches however, multiprocessors have not been selected yet for critical computations.
- Early long-life systems employed hardware masking redundancy on uniprocessors. Hardware handled all of the various stages of a fault. The recent trend has been to multicomputers and standby configurations using hardware for error-detection while isolation and recovery are performed by software.

While there have been a rich variety of fault-tolerant architectures conceived and constructed in the past, there is likely to be a convergence in structures and approaches. Just as the economics of volume manufacturing make standards such as microprocessors, buses, and operating systems, the basic building blocks of contemporary computer architectures, in the not too distant future fault-tolerant architectures will also approach standardization. Several microprocessors, such as the Motorola 88000, already provide the necessary logic to build duplicated systems. Error detection and correction codes have proven very effective for regular logic such as memories and in the not too distant future memory chips may have built-in support for error detection and correcting codes. Finally, with the ever-increasing dominance of transient and intermittent failures, retry mechanisms will be built into all levels of the system as the major error-recovery mechanism. As peripheral devices such as disks shrink in size, we shall see duplication and coding techniques extending beyond the central processor.

Fault tolerance is no longer an exotic engineering discipline, rather it is becoming as fundamental to computer design as logic synthesis. Designs will be compared and contrasted not only by their cost, power consumption, and performance but also by their reliability and ability to tolerate failures.

ACKNOWLEDGMENT

The author would like to thank the reviewers for many thoughtful comments that improved the accuracy of the manuscript.

REFERENCES

[1] Daniel P. Siewiorek and Robert S. Swarz, *Reliable Computer Systems: Design and Evaluation*. Bedford, MA: Digital Press, 1992.
[2] Jean-Claude Laprie, "Dependable computing and fault-tolerance: concepts and terminology," in *Fifteenth Annual Int. Symp. Fault-Tolerant Computing*, IEEE Computer Society, Ann Arbor, MI, pp. 2–11, June 1985.
[3] A. Avizienis, "Architecture of fault-tolerant computing systems," in *Digest Fifth Int. Symp. Fault-Tolerant Computing*, IEEE Computer Society, Paris, France, 1975, pp. 3–16.
[4] D. P. Siewiorek and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*. Bedford, MA: Digital Press, 1982.
[5] T. Anderson and R. Kerr, "Recovery blocks in action: A system supporting high reliability," in *Proc. 2nd Int. Conf. Software Eng.*, San Francisco, CA, Oct. 1976, pp. 447–457.
[6] T. Anderson and P. A. Lee, *Fault Tolerance Principles and Practices*. London, U.K.: Prentice-Hall, 1981.
[7] A. Avizienis and L. Chen, "On the implementation of N-version programming for software fault tolerance during execution," in *Proc. COMPDSAC77*, 1977, pp. 149–155.
[8] A Avizienis and J. P. J. Kelly, "Fault-tolerance by design diversity: concepts and experiments," *Computer*, vol. 17, pp. 67–80, Aug. 1984.
[9] J. C. Knight, N. G. Leveson, and L. St. Jean, "A large-scale experiment in N-version programming," in *15th Int. Conf. Fault-Tolerant Computing, (FTCS-15)*, IEEE Computer Socity, Ann Arbor, MI, 1985, pp. 135–139.
[10] D. P. Siewiorek, M. Y. Hsiao, D. Rennels, J. Gray, and T. Williams, "Ultradependable architectures," *Ann. Rev. Comput. Sci.*, vol. 4, pp. 503–515, 1989–1990.
[11] W. Toy, "Fault-tolerant design of local ESS processors," *Proc. IEEE*, vol. 66, pp. 1126–1145, Oct. 1978.
[12] Ali, Bellcore, private communication, 1986.
[13] J. Gray, Tandem, private communication, 1985, 1987.
[14] D. P. Siewiorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples*. New York: McGraw-Hill, 1982.
[15] D. A. Rennels, "Distributed fault-tolerant computer systems," *Computer*, vol. 13, no. 3, pp. 55–65, Mar. 1980.
[16] E. F. Moore and C. E. Shannon, "Reliable circuits using less reliable relays," *J. Franklin Inst.*, vol. 262, pp. 191–208, Sept. 1956.
[17] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies*, C. E. Shannon and J. McCarthy, Eds. Princeton, NJ: Princeton University Press, 1956, pp. 43–98.
[18] R. H. Wilcox and W. C. Mann, Eds. *Redundancy Techniques for Computer Systems*. Washington, D.C.: Spartan Books, 1962.
[19] A. Avizienis, "Design of fault tolerant computers," in *FJCC AFIPS Conf. Proc.*, vol. 31, AFIPS Press, Montvale, NJ, 1967, pp. 732–743.
[20] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*. Reading, MA: Addison-Wesley, 1989.
[21] B. E. Ossfeldt and I. Jonsson, "Recovery and diagnostics in the central control of the AXE switching system," *IEEE Trans. Computers*, vol. C-29, pp. 482–491, June 1980.
[22] M. Y. Hsiao, "A class of optimal minimum odd-weight column SEC-DED codes," *IBM J. Res. Develop.*, vol. 14, pp. 395–401, 1970.
[23] D. T. Tang and R. T. Chien, "Coding for error control," *IBM Syst. J.*, vol. 8, no. 1, pp. 48–85, 1969.
[24] W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes*. Cambridge, MA: MIT Press, 1972.
[25] E. R. Berlekamp, *Algebraic Coding Theory*. New York: McGraw-Hill, 1968.
[26] F. J. McWilliams and N. T. A. Sloan, *The Theory of Error-Correcting Codes*. New York: North-Holland, 1978.
[27] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," (special issue on Coding and Error Control,) *IBM J. Res. Devel.*, pp. 124–134, 1984.
[28] M. Morganti, G. Coppadoro, and S. Ceru, "UDET 7116—common control for PCM telephone exchange diagnostic software design and availability evaluation," in *Digest Eighth Int. Fault-Tolerant Computing Symp.*, IEEE Computer Society, Toulouse, France, 1978, pp. 16–23.
[29] H. Ihara, K. Fukuoka, Y. Kubo, and S. Yokota, "Fault-tolerant computer system with three symmetric computers," *Proc. IEEE*, vol. 66, pp. 1160–1177, Oct. 1978.
[30] W. J. Wachter, "System malfunction detection and correction," in *Digest Fifth Int. Fault-Tolerant Computing Symp.*, IEEE Computer Society, Paris, France, 1975, pp. 196–201.
[31] B. P. Randell, P. A. Lee, and P. C. Treleavan, "Reliability issues in computer system design," *Computing Surveys*, vol. 10, no. 2, pp. 123–165, June 1978.
[32] D. L. Droulette, "Recovery through programming system/360-system/370," in *SJCC AFIPS Conf. Proc.*, vol. 38, AFIPS Press, Montvale, NJ, 1971, pp. 467–476.
[33] D. DeAngelis and J. A. Lauro, "Software recovery in the fault-tolerant spaceborne computer," in *Digest Sixth Int. Fault-Tolerant Computing Symp.*, IEEE Computer Society, Pittsburgh, PA, 1976, pp. 143–148.
[34] F. J. O'Brien, "Rollback point insertion strategies," in *Digest Sixth Int. Fault-Tolerant Computing Symp.*, IEEE Computer Society, Pittsburgh, PA, 1976, pp. 138–142.
[35] J. J. Stiffler, "Architectural design for near-100% fault coverage," in *Digest Sixth Int. Fault-Tolerant Computing Symp.*, IEEE Computer Society, Pittsburgh, PA, 1976.
[36] C. Meraud, F. Browarys, and G. Germain, "Automatic rollback techniques of the COPRA computer," in *Digest Sixth Int. Fault-Tolerant Computing Symp.*, IEEE Computer Society, Pittsburgh, PA, 1976, pp. 23–31.
[37] C. Meraud, F. Browarys, J. P. Queille, and G. Germain, "Hardware and software design of the fault-tolerant computer COPRA," in *Digest Ninth Int. Fault-Tolerant Computing Symp.*, IEEE Computer Society, Madison, WI, p. 167.
[38] A. Avizienis, G. C. Gilley, F. P. Mathur, D. A. Rennels, J. A. Rohr, and D. K. Rubin, "The STAR (self-testing and repairing) computer: An investigation on the theory and practice of fault-tolerant computer design," *IEEE Trans. Computers*, vol. C-20, pp. 1312–1321, Nov. 1971.
[39] A. Borg, J. Baumbach, and S. Glazer, "A message system supporting fault tolerance," in *Proc. Ninth ACM Symp. Operating Systems Principles*, Bretton Woods. (Also published in the *ACM Op. System. Rev.*, vol. 17, no. 5, Oct. 10–13, 1983.)
[40] H. Wyle and G. J. Burnett, "Some relationships between failure detection probability and computer system reliability," in *FJCC, AFIPS Conf. Proc.*, vol. 31, Montvale, NJ, Academic Press, 1967, pp. 745–756.
[41] W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Reliability modeling techniques for self-repairing computer systems," in *Proc. 24th National Conf. ACM*, ACM 1969, pp. 295–309.
[42] J. P. Eckert, Jr., J. R. Weiner, H. F. Welsh, and H. F. Mitchell. "The UNIVAC system," *AIEE-IRE Conf.*, Dec. 1951, pp. 6–16.
[43] W. C. Carter, H. C. Montgomery, R. J. Preiss, and H. J. Reinheimer, "Design of serviceability features for the IBM system/360," *IBM J. Res. Develop.*, vol. 8, no. 2, pp. 115–126, Apr. 1964.
[44] M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow, "Reliability, availability, and serviceability of IBM computer systems: A quarter century of progress," *IBM J. Res. Develop.*, vol. 25, no. 5, pp. 454–465, Sept. 1981.
[45] D. C. Bossen and M. Y. Hsiao, "Model for transient and permanent error-detection and fault isolation," *IBM J. Res. Develop.*, vol. 26, no. 1, pp. 67–77, Jan. 1982.
[46] N. N. Tendolkar and R. I. Swan, "Automated diagnostic methodology for IBM 3081 processor complex," *IBM J. Res. Develop.*, vol. 26, no. 1, pp. 78–88, Jan. 1982.
[47] W. Toy and L. Toy, "Fault-tolerant design of electronic switching systems," in *Reliable Computer Systems: Design and Evaluation*. Bedford, MA: Digital Press, 1992.
[48] W. Keister, R. W. Ketchledge, and C. A. Lovell. "Morris electronic telephone exchange," *Proc. Inst. Elec. Eng.*, vol. 107, no. 20, pp. 257–263, 1964.
[49] C. F. Ault, L. E. Gallaher, T. S. Greenwood, and D. C. Koehler. "No. 1 ESS program store," *Bell Syst. Tech. J.*, vol. 43, pp. 2097–2146, Sept. 1964.
[50] W. B. Cagle, R. S. Menne, R. S. Skinner, R. E. Staehler, and M. D. Underwood, "No. 1 ESS logic circuits and their application to the

design of the central control," *Bell Syst. Tech. J.*, vol. 43, no. 5, Pt. 1, pp. 2055–2095, Sept. 1964.
[51] R. W. Downing, J. S. Nowak, and L. S. Toumenoksa, "No. 1 ESS maintenance plan," *Bell Syst. Tech. J.*, vol. 43, no. 5, pt. 1, pp. 1961–2019, Sept. 1964.
[52] W. Keister, R. W. Ketchledge, and H. E. Vaughan, "No. 1 ESS system organization and objectives," *Bell Syst. Tech. J.*, vol. 43, no. 5, pt. 1, pp. 1831–1844, Sept. 1964.
[53] T. E. Browne, T. M. Quinn, W. N. Toy, and J. E. Yates, "No. 2 ESS control unit system," *Bell Syst. Tech. J.*, vol. 48, no. 2, pp. 443–476, Oct. 1969.
[54] A. E. Spencer and F. S. Vigilante, "No. 2 ESS—System organization and objectives," *Bell Syst. Tech. J.*, vol. 48, pp. 2607–2618, Oct. 1969.
[55] P. J. Kennedy and T. M. Quinn, "Recovery strategies in the No. 2 ESS," in *Dig. Second Int. Fault-Tolerant Computing Symp.*, IEEE, Boston, MA, 1972.
[56] W. O. Flechenstein, "Bell system ESS family—present and future," in *ISS Record*, Munich, Germany, 1974.
[57] P. D. Mandigo, "No. 2B ESS: New features for a more efficient processor," *Bell Labs Rec.*, vol. 54, no. 11, pp. 304–309, Dec. 1976.
[58] J. S. Nowak, "No. 1A ESS-A new high capacity switching system," in *Int. Switching Symp. Record*, Japan, 1976.
[59] R. E. Staehler and R. J. Watters, "1A Processor—An ultra-dependable common control," in *Int. Switching Symp. Record*, Japan, 1976.
[60] T. F. Storey, "Design of a microprogram control for a processor in an electronic switching system," *Bell Syst. Tech. J.*, no. 2, pp. 183–232, Feb. 1976.
[61] C. F. Ault, J. H. Brewster, T. S. Greenwood, R. E. Haglund, W. A. Read, and M. W. Rolund, "1A processor-memory systems," *Bell Syst. Tech. J.*, vol. 56, no. 2, pp. 181–205, Feb. 1977.
[62] P. W. Bowman, M. R. Diebman, F. M. Gaety, R. F. Krantzmann, E. H. Stredde, and R. J. Watters, "1A processor—Maintenance software," *Bell Syst. Tech. J.*, vol. 56, pp. 225–287, Feb. 1977.
[63] A. H. Budlong, B. G. DeLiegish, I. M. Neville, J. S. Nowak, J. L. Quinn, and R. W. Wendloud, "1A processor—Control system," *Bell Syst. Tech. J.*, vol. 56, no. 2, pp. 135–179, Feb. 1977.
[64] J. J. Kulzer, "Systems reliability: A case study of No. 4 ESS," in *System Security and Reliability*, Maidenhead, Bershire, England, Infotech International Ltd., pp. 186–188, 1977.
[65] R. E. Staehler, "1A Processor—Organization and objective," *Bell Syst. Tech. J.*, vol. 56, no. 2, pp. 119–134, Feb. 1977.
[66] J. R. Becker, J. G. Cheoalier, R. K. Eisenhart, J. H. Forster, A. W. Fulton, W. L. Harrod, "1A processor—Technology and physical design," *Bell Syst. Tech. J.*, vol. 56, pp. 207–236, Feb. 1977.
[67] J. H. Wallace, and W. W. Barnes, "Designing for ultra-high availability: The Unix RTR operating system," *Computer*, vol. 17, no. 8, pp. 31–39, 1984.
[68] R. A. Maxion, Daniel P. Siewiorek, and Steven A. Elkind, "Techniques and architectures for fault-tolerant computing," *Annual Review of Computer Science*, vol. 2, pp. 469–520, Annual Reviews, Inc., 1987.
[69] J. F. Bartlett, "A 'NonStop' kernel," in *Hawaii Int. Conf. System Sciences*, Honolulu, Hawaii, 1978, pp. 103–119.
[70] J. F. Bartlett, "A 'NonStop' kernel," in *ACM 8th Symp. Operating System Principles*, Pacific Grove, CA, 1981, pp. 22–29.
[71] J. A. Katzman, "System architecture for NonStop computing," in *CompCon*, pp. 77–80, 1977.
[72] J. A. Katzman, "A fault-tolerant computing system," Tandem Computers, Inc., Cupertino, CA, 1977.
[73] Synapse Transaction Processing System Overview, Synapse Computer Corporation, Milpitas, CA 1983.
[74] R. Gostanian, "The Auragen System 4000," *IEEE Q. Bull. Database Eng.* (special issue, Highly Available Systems), vol. 6, pp. 62–72, 1983.
[75] S. Webber, "The Stratus architecture," in *Reliable Computer Systems: Design and Evaluation*. Bedford, MA: Digital Press, 1992.
[76] D. P. Siewiorek and D. Johnson, "A design methodology for high reliability systems: The Intel 432," in *The Theory and Practice of Reliable System Design*, Ch. 18. Bedford, MA: Digital Press, 1982.
[77] D. Johnson, "The Intel 432: a VLSI architecture for fault-tolerant computer systems," *IEEE Computer*, vol. 17, pp. 40–48, 1984.
[78] D. P. Siewiorek, V. Kini, H. Mashburn, S. McConnel, and M. Tsao, "A case study of C.mmp, Cm*, and C.vmp: part 1–Experience with fault-tolerance in multiprocessor systems," *Proc. IEEE*, vol. 66, pp. 1178–1199, Oct. 1978.
[79] M. M. Dickenson, J. B. Jackson, and G. C. Randa, "Saturn V launch vehicle digital computer and data adapter," in *FJCC AFIPS Conf. Proc.*, vol. 26, 1964, pp. 501–516.
[80] A. L. Hopkins, Jr., T. B. Smith, III, J. Lala, "FTMP—A Highly reliable fault-tolerant multiprocessor for aircraft," *Proc. IEEE*, vol. 66, pp. 1221–1239, Oct. 1978.
[81] J. H. Wensley, "SIFT software implemented fault tolerance," in *FJCC AFIPS Conf. Proc.*, AFIPS Press, 1972, pp. 243–253.
[82] J. F. Wensley, M. W. Green, K. N. Levitt, and R. E. Shostak, "The design, analysis, and verification of the SIFT fault tolerant system," in *Proc. 2nd Int. Conf. Software Engineering*, IEEE Computer Society, 1976, pp. 458–469.
[83] J. F. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," in *Proc. IEEE Computers*, Oct. 1978, pp. 1240–1255.
[84] J. H. Wensley, "August systems industrial control computers," in *Resilient Computing Systems*, T. Anderson, Ed. New York: Wiley, 1985, pp. 232–245.
[85] F. P. Mathur and A. Avizienis, "Reliability analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redundancy with self-repair," in *SJCC, AFIPS Conf. Proc.*, vol. 36, Montvale, NJ, AFIPS Press, 1970, pp. 375–383.
[86] C. P. Jones, "Automatic fault protection in the Voyager Spacecraft," *AAIA Paper. 79-1919*, Calif. Inst. Technol., Jet Propulsion Lab. (see also ch. 15 in reference [4].)
[87] R. W. Kocsis, "Galileo orbiter fault protection system," in *Reliable Computer Systems: Design and Evaluation*. Bedford, MA: Digital Press, 1992.
[88] D. P. Siewiorek and R. W. Swarz, "RAMP in the VAX Family," in *Reliable Computer Systems: Design and Evaluation*. Bedford, MA: Digital Press, 1992.
[89] B. R. Borgerson, M. L. Hanson, and P. A. Hartley, "The evolution of the Sperry Univac 1100 Series: A history, analysis and projection," *Commun. ACM 1*, pp. 25–43, Jan. 1978.
[90] D. J. Kunshier and D. R. Mueller, "Support processor based system fault recovery," in *Digest of Tenth Int. Symp. Fault Tolerant Computing*, IEEE Computer Society, Kyoto, Japan, 1980, pp. 297–301.
[91] L. A. Boone, H. L. Liebergot, and R. M. Sedmak, "Availability, reliability, and maintainability aspects of the Sperry Univac 1100/60," in *Digest of Tenth Int. Fault-Tolerant Computing Symp.*, Kyoto, Japan, IEEE Computer Society, 1980, pp. 3–8.
[92] R. R. Everett, C. A. Zraket, and H. D. Bennington, "SAGE: A data-processing system for air defense," in *Proc. EJCC*, 1957, pp. 148–155.
[93] F. E. Heart, S. M. Ornstein, W. R. Crowther, and W. B. Barker, "A new minicomputer/multiprocessor for the ARPA network," in *AFIPS Conf. Proc.*, vol. 42, Montvale, NJ, AFIPS Press, 1973, pp. 529–537.
[94] S. M. Ornstein, W. R. Crowther, M. F. Kraley, R. D. Bressler, A. Michel, and F. E. Heart. "Pluribus—A reliable multiprocessor," in *AFIPS Conf. Proc.*, Montvale, NJ, AFIPS Press, 1975, pp. 551–559.
[95] F. E. Heart, S. M. Ornstein, W. R. Crowther, W. B. Barker, M. F. Kraley, R. D. Bressler, and A. Michel, "The Pluribus multiprocessor system," in *Multiprocessor Systems: Infotech State of the Art Report*, Maidenhead, England, Infotech International Ltd., pp. 307–330, 1976.
[96] R. T. Gudz, "Application of the Pluribus multiprocessor in a distributed data collect and processing network," in *Conf. Recordings, OCEANS 77*, 1977.
[97] D. Katsuki, E. S. Elsam, W. F. Mann, E. S. Roberts, and E. W. Wolf, "Pluribus—An operational fault-tolerant multiprocessor," *Proc. IEEE*, vol. 66, pp. 1146–1159, Oct. 1978.
[98] P. A. Bernstein, "Sequoia: A fault-tolerant tightly coupled multiprocessor for transaction processing," *Computer*, vol. 21, no. 2, pp. 37–45, Feb. 1988.
[99] J. H. Lala, "A Byzantine resilient fault tolerant computer for nuclear power plant application," in *IEEE FTCS-26 Digest of Papers*, pp. 383–343, Vienna, Austria, July 1–4, 1986.
[100] A. E. Cooper and W. T. Chow, "Development of on-board space computer systems," *IBM J. Res. Develop.*, vol. 20, no. 1, pp. 5–19, 1976.
[101] J. R. Sklaroff, "Redundancy management technique for space shuttle computers," *IBM J. Res. Develop.*, vol. 20, no. 1, pp. 20–28, Jan. 1976.
[102] "Velocity, altitude regimes to push computer limits," *Aviation Week & Space Technology*, pp. 49–51, Apr. 1981.

[103] L. S. Alger and J. H. Lala, "Performance evaluation of a realtime fault tolerant distributed system," in *Proc. Twenty-Third Annual Hawaii Int. Conf. System Sciences,* vol. 1, Kailua-Kona, Hawaii, Jan. 2–5, 1990, pp. 278–281.

Daniel P. Siewiorek (Fellow, IEEE) was born in Cleveland, OH, on June 2, 1946. He received the B.S. degree in electrical engineering (summa cum laude) from the University of Michigan, Ann Arbor, in 1968, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1969 and 1972, respectively.

He is a professor in the School of Computer Science and the CIT Department of Electrical and Computer Engineering at Carnegie Mellon University, where he helped to initiate and guide the Cm* project that culminated in an operational 50-processor multiprocessor system. He has been a key contributor to the dependability design of over two dozen commercial computing systems. His current research interest include computer architecture, reliability modeling, fault tolerant computing, modular design, and design automation. He has conducted research and served as a consultant to several commercial and government organizations, including Digital Equipment Corporation, Jet Propulsion Laboratory, The Naval Research Laboratory, Research Triangle Institute, and United Technologies Corporation. Dr. Siewiorek has published over 200 technical papers and five books, including "The Theory and Practice of Reliable System Design."

Dr. Siewiorek was recognized with an Honorable Mention Award as Outstanding Young Electrical Engineer in 1977, given by Eta Kappa Nu (National Electrical Engineering Honorary Society); Dr. Siewiorek was elected an IEEE Fellow in 1981 for "contributions to the design of modular computing systems;" and has served as chairman of the IEEE Technical Committee on Fault-Tolerant Computing. He was also awarded the Frederick Emmons Terman Award by the American Society for Engineering Education for Outstanding Young Electrical Engineering Educator and in 1988 he received the Eckert-Mauchly Award given jointly by the ACM and IEEE Computer Society for his outstanding contributions to the field of computer architecture. He is a member of the ACM, Tau Beta Pi, Eta Kappa Nu, and Sigma Xi.